3

Structures That Control Flow


VideoNote
Relational and Logical Operators

3.1 Relational and Logical Operators

In Chapter 1, we discussed the two logical programming constructs *decision* and *loop*. In this chapter, we learn how to implement decision and loop structures. In order to make decisions (and often in order to control loops), we must specify a condition that determines the course of action.

A **condition** (or **Boolean expression**) is an expression involving relational operators (such as < and >=) and logical operators (such as *and*, *or*, and *not*). ASCII values determine the order used to compare strings with relational operators. A condition evaluates to either True or False (referred to as the **truth value** of the condition). True and False are reserved words.

ASCII Values

Each of the 47 keys in the center typewriter portion of the keyboard can produce two characters, for a total of 94 characters. Adding 1 for the character produced by the space bar makes 95 characters. Associated with these characters are numbers ranging from 32 to 126. These values, called the ASCII values of the characters, are given in Appendix A. Table 3.1 shows a few ASCII values.

TABLE 3.1 A few ASCII values.

32	(space)	48	0	66	B	122	z
33	!	49	1	90	Z	123	{
34	"	57	9	97	a	125	}
35	#	65	A	98	b	126	~

The ASCII standard also assigns characters to some numbers above 126. Table 3.2 shows a few of the higher ASCII values.

TABLE 3.2 A few higher ASCII values.

162	¢	177	±	181	µ	190	¾
169	©	178	²	188	¼	247	÷
176	°	179	³	189	½	248	ø

If *n* is a nonnegative number, then

```
chr(n)
```

is the single-character string consisting of the character with ASCII value *n*. If *str* is any single-character string, then

```
ord(str)
```

is the ASCII value of the character. For instance, the statement

```
print(chr(65))
```

displays the letter A, and the statement

```
print(ord('A'))
```

displays the number 65.

Concatenation can be used with chr to obtain strings using the higher ASCII characters. For instance, the statement

```
print("32" + chr(176) + " Fahrenheit")
```

displays `32° Fahrenheit`.

Relational Operators

The relational operator *less than* (<) can be applied to numbers, strings, and other objects. The number a is said to be less than the number b if a lies to the left of b on the number line. For instance, $2 < 5$, $-5 < -2$, and $0 < 3.5$.

The string a is said to be less than the string b if a precedes b when using the ASCII table to order their characters. Digits precede uppercase letters, which precede lowercase letters. Two strings are compared character by character (working from left to right) to determine which string should precede the other. Thus, "cat" < "dog", "cart" < "cat", "cat" < "catalog", "9W" < "bat", "Dog" < "cat", and "sales_99" < "sales_retail". This type of ordering is called **lexicographical ordering**. Table 3.3 shows the different relational operators and their meanings.

TABLE 3.3 Relational operators.

Python Notation	Numeric Meaning	String Meaning
==	equal to	identical to
!=	not equal to	different from
<	less than	precedes lexicographically
>	greater than	follows lexicographically
<=	less than or equal to	precedes lexicographically or is identical to
>=	greater than or equal to	follows lexicographically or is identical to
in		substring of
not in		not a substring of

Example 1 Relational Operators Determine whether each of the following conditions evaluates to True or False.

(a) 1 <= 1
(b) 1 < 1
(c) "car" < "cat"
(d) "Dog" < "dog"
(e) "fun" in "refunded"

SOLUTION

(a) True. The notation <= means "less than *or* equal to." That is, the condition is true provided either of the two situations holds. The second one (equal to) holds.

(b) False. The notation < means "strictly less than" and no number can be strictly less than itself.

(c) True. The characters of the strings are compared one at a time working from left to right. Because the first two characters match, the third character determines the order.

(d) True. Because uppercase letters precede lowercase letters in the ASCII table, the first character of "Dog" precedes the first character of "dog".

(e) True. The string "fun" is "refunded"[2:5], a substring of "refunded".

Conditions can also involve variables, numeric operators, and functions. To determine whether a condition is true or false, first evaluate the numeric or string expressions and then decide if the resulting assertion is true or false.

Example 2 **Relational Operators** Suppose the variables *a* and *b* have values 4 and 3, and the variables *c* and *d* have values "hello" and "bye". Are the following conditions true or false?

(a) (a + b) < (2 * a)

(b) (len(c) − b) == (a/2)

(c) c < ("good" + d)

SOLUTION

(a) True. The value of *a* + *b* is 7 and the value of 2 * *a* is 8. Since 7 < 8, the condition is true.

(b) True, because the value of len(c) − b is 2, the same as (a/2).

(c) False. The condition "hello" < "goodbye" is false, since *h* follows *g* in the alphabet.

An **int** can be compared to a **float**. Otherwise, values of different types cannot be compared. For instance, a string cannot be compared to a number.

The relational operators can be applied to lists or tuples. In order for two lists or two tuples to be equal, they must have the same length and corresponding items must have the same value. The truth value of the condition is determined by comparing successive corresponding items until the two items differ (or cannot be compared) or until one of the sequences runs out of items. The first pair of items that have different values determine the truth value of the condition. If one of the sequences runs out of items and all items pairs match, then the shorter sequence is said to be the lesser of the two. Some examples of comparisons having truth value **True** are as follows:

```
[3, 5] < [3, 7]
[3, 5] < [3, 5, 6]
[3, 5, 7] < [3, 7, 2]
[7, "three", 5] < [7, "two", 2]
```

When the **in** operator is applied to a list or tuple, it should be taken to mean *is an item of*. Two true examples are as follows:

```
'b' in ['a', 'b', 'c']
'B' not in ('a', 'b', 'c')
```

Sorting the Items in a List

The items in a list where every pair of items can be compared can be ordered with the sort method. The statement

```
list1.sort()
```

changes *list1* to a list having the same items, but in ascending order either numerically or lexicographically as appropriate.

Example 3 Sort a List The following program illustrates how Python orders two simple lists.

```
list1 = [6, 4, -5, 3.5]
list1.sort()
print(list1)
list2 = ["ha", "hi", 'B', '7']
list2.sort()
print(list2)
```

[Run]

```
[-5, 3.5, 4, 6]
['7', 'B', 'ha', 'hi']
```

Example 4 Sort a List The following program illustrates how Python orders the items in a complicated list of strings. ***Note:*** chr(177) is the ± character and chr(162) is the ¢ character.

```
list1 = [chr(177), "cat", "car", "Dog", "dog", "8-ball", "5" + chr(162)]
list1.sort()
print(list1)
```

[Run]

```
['5¢', '8-ball', 'Dog', 'car', 'cat', 'dog', '±']
```

Example 5 Sort a List The following program orders the items in a list of tuples.

```
monarchs = [("George", 5), ("Elizabeth", 2), ("George", 6), ("Elizabeth", 1)]
monarchs.sort()
print(monarchs)
```

[Run]

```
[('Elizabeth', 1), ('Elizabeth', 2), ('George', 5), ('George', 6)]
```

Logical Operators

Programming often requires more complex conditions than those considered so far. For instance, suppose we would like to state that the value of the variable *str1* is a string of length 10 and contains the substring "gram". The proper Python condition is

```
(len(str1) == 10) and ("gram" in str1)
```

This condition is a combination of the condition `(len(str1) == 10)` and the condition `("gram" in str1)` with the logical operator **and**.

The three main logical operators are the reserved words **and**, **or**, and **not**. Conditions that use these operators are called **compound conditions**. If *cond1* and *cond2* are conditions, then the compound condition

cond1* and *cond2

is true if both of the conditions are true. Otherwise, it is false. The compound condition

cond1* or *cond2

is true if either (or both) of the two conditions are true. Otherwise, it is false. The compound condition

not *cond1*

is true if the condition is false, and is false if the condition is true.

Example 6 **Logical Operators** Suppose the variable *n* has value 4 and the variable *answ* has value "Y". Determine whether each of the following conditions evaluates to **True** or **False**.

(a) (2 < n) and (n < 6)
(b) (2 < n) or (n == 6)
(c) not (n < 6)
(d) (answ == "Y") or (answ == "y")
(e) (answ == "Y") and (answ == "y")
(f) not (answ == "y")
(g) ((2 < n) and (n == 5 + 1)) or (answ == "No")
(h) ((n == 2) and (n == 7)) or (answ == "Y")
(i) (n == 2) and ((n == 7) or (answ == "Y"))

SOLUTION

(a) **True**, because the conditions (2 < 4) and (4 < 6) are both true.
(b) **True**, because the condition (2 < 4) is true. The fact that the condition (4 == 6) is false does not affect the conclusion. The only requirement is that at least one of the two conditions be true.
(c) **False**, because (4 < 6) is true.
(d) **True**, because the first condition becomes ("Y" == "Y") when the value of *answ* is substituted for *answ*.
(e) **False**, because the second condition is false. Actually, this compound condition is false for any value of *answ*.
(f) **True**, because ("Y" == "y") is false.
(g) **False**. In this logical expression, the compound condition ((2 < n) and (n == 5 + 1)) and the simple condition (answ == "No") are joined by the logical operator or. Because both these conditions are false, the total condition is false.

(h) True, because the condition following *or* is true.

(i) False, because the first condition is false. (Comparing (h) and (i) shows the necessity of using parentheses to specify the intended grouping.)

Short-Circuit Evaluation

When Python encounters the compound condition `(cond1 and cond2)`, it first evaluates `cond1`. If `cond1` is false, Python realizes that the compound condition is false and therefore does not bother to evaluate `cond2`. Similarly, when Python encounters the compound condition `(cond1 or cond2)`, it first evaluates `cond1`. If `cond1` is true, Python realizes that the compound condition is true and therefore does not bother to evaluate `cond2`. This process is called **short-circuit evaluation**.

Some programming languages evaluate both parts of a compound condition before assigning a value to the compound condition. If so, evaluation of the condition

```
(number != 0) and (m == (n / number))
```

will cause the program to crash and display an error message when *number* has the value 0. However, due to short-circuit evaluation, the evaluation of this compound condition will never cause a problem in Python.

Short-circuit evaluation sometimes improves the performance of a program. Such can be the case, for instance, when the evaluation of `cond2` is time-consuming.

The *bool* Data Type

A statement of the form

```
print(condition)
```

will display either True or False. The objects True and False are said to have **Boolean data type** or to be of data type bool. The following lines of code display False:

```
x = 5
print((3 + x) < 7)
```

The following lines of code display True:

```
x = 2
y = 3
var = x < y
print(var)
```

The answer to part (i) of Example 6 can be confirmed to be False by executing the following lines of code:

```
n = 4
answ = "Y"
print((n == 2) and ((n == 7) or (answ == "Y")))
```

Three Methods That Return Boolean Values

If *str1* and *str2* are strings, then the condition

```
str1.startswith(str2)
```

has the value True if and only if *str1* begins with *str2*, and the condition

```
str1.endswith(str2)
```

has the value True if and only if *str1* ends with *str2*.

For instance, the following two conditions are true:

```
"fantastic".startswith("fan")
"fantastic".endswith("stic")
```

If *var1* has the value "fantastic" and *var2* has the value "Fant", then the following two conditions are false:

```
var1.startswith(var2)
"elephant".endswith(var2)
```

If *item* is a literal or variable, then a condition of the form

```
isinstance(item, dataType)
```

has the value True if and only if the value of *item* has the specified data type, where *dataType* is any data type (such as int, float, str, bool, list, or tuple).

For example, the condition `isinstance("32", int)` has the value False and the condition `isinstance(32, int)` has the value True.

Table 3.4 shows several other string methods that return Boolean values. In the table, assume that *str1* is not the empty string. Each of the methods in the table returns False when *str1* is the empty string.

TABLE 3.4 Methods that return either True or False.

Method	Returns True when
str1.isdigit()	all of *str1*'s characters are digits
str1.isalpha()	all of *str1*'s characters are letters of the alphabet
str1.isalnum()	all of *str1*'s characters are letters of the alphabet or digits
str1.islower()	*str1* has at least 1 alphabetic character and all of its alphabetic characters are lowercase
str1.isupper()	*str1* has at least 1 alphabetic character and all of its alphabetic characters are uppercase
str1.isspace()	*str1* contains only whitespace characters

Simplifying Conditions

Lists or tuples can sometimes be used to simplify long compound conditions containing logical operators. For instance, the compound condition

```
(state == "MD") or (state == "VA") or (state == "WV") or (state == "DE")
```

can be replaced with the condition

```
state in ["MD", "VA", "WV", "DE"]
```

Sometimes compound conditions involving inequalities can be written in a clearer form. For instance, the condition

```
(x > 10) and (x <= 20)
```

can be replaced with the condition

```
10 < x <= 20
```

and the condition

```
(x <= 10) or (x > 20)
```

can be replaced with the condition

```
not(10 < x <= 20)
```

Two principles of logic, known as **De Morgan's Laws**, are as follows:

`not(`*`cond1`* `and` *`cond2`*`)` is the same as `not(`*`cond1`*`) or not(`*`cond2`*`)`

`not(`*`cond1`* `or` *`cond2`*`)` is the same as `not(`*`cond1`*`) and not(`*`cond2`*`)`

De Morgan's Laws can be applied from left to right or from right to left. For instance, according to De Morgan's Laws, the compound condition

```
not((temperature >= 80) and (humidity <= 60))
```

is the same as

```
(temperature < 80) or (humidity > 60))
```

and the compound condition

```
not(len(word) == 5) and not(word.startswith('A'))
```

is the same as

```
not((len(word) == 5) or (word.startswith('A')))
```

Comments

1. A condition involving numeric variables is different from an algebraic identity or inequality. The assertion (a + b) < (2 * a) considered in Example 2 is not a valid algebraic inequality because it isn't true for all values of *a* and *b*. When encountered in a Python program, however, it will be considered true if it is true for the current values of the variables.
2. A common error is to replace the condition (not (n < m)) with the condition (n > m). The correct replacement is (n >= m).
3. The condition "three" == 3 evaluates to False, but the condition "three" < 3 triggers a Traceback error.
4. A common error is to use a single equal sign in a condition where a double equal sign is required.
5. The sort method cannot be used in an assignment statement. For instance, the statement `list2 = list1.sort()` is not valid because sort does not return a value; it just reorders the items in place. It can be replaced with the following pair of statements:

   ```
   list1.sort()
   list2 = list1
   ```

6. Since the words and, or, not, True, and False are reserved words, they are colorized orange by IDLE.

Practice Problems 3.1

1. Does the condition `"Hello " == "Hello"` evaluate to True or False?
2. Explain why (27 > 9) evaluates to True, whereas ("27" > "9") evaluates to False.
3. Complete Table 3.5.

TABLE 3.5 Truth values of logical operators.

cond1	cond2	cond1 and cond2	cond1 or cond2	not cond2
True	True	True		
True	False		True	
False	True			False
False	False			

4. Consider Example 5. Suppose that *monarchs* had been a tuple instead of a list. Is there a way to order the items in *monarchs* even though tuples do not have a sort method?
5. What is displayed by the statement `print("Hello World".isalpha())`?
6. What is the difference between = and ==?

EXERCISES 3.1

In Exercises 1 through 8, determine the output displayed.

1.
```
print(chr(42)*5)
```
2.
```
print('Py'+chr(116)+'ho'+chr(110))
```
3.
```
print("The upper case of letter g is " + chr((ord('g') -  ord('a')) +
ord('A') ) + '.')
```
4.
```
print(chr(ord('B'))) # The ASCII value of B is 66
```
5.
```
list1 = [17, 3, 12, 9, 10]
list1.sort()
print("Minimum:", list1[0])
print("Maximum:", list1[-1])
```
6.
```
list1 = [17, 3, 12, 9, 10]
list1.sort()
print("Spread:", list1[-1] - list1[0])
```
7.
```
letter = 'D'
print(letter + " is 4 positions before " + chr(ord(letter) + 4) +
" alphabetically.")
```
8.
```
letter = 'D'
spread = ord('a') - ord('A')
print(chr(ord(letter) + spread))
```

In Exercises 9 through 20, assume the value of a is 1 and the value of b is 1.5, and determine whether the condition evaluates to True or False. Then, use a print function to confirm your answer.

9. `3 * a == 2 * b`
10. `((5 - a) * b) < 7`
11. `b <= 3`
12. `a ** b == b ** a`
13. `a ** (5 - 2) > 7`
14. `3e-2 < .01 * a`
15. `(a < b) or (b < a)`
16. `(a * a < b) or not(a * a < a)`
17. `not((a < b) and (a < (b + a)))`
18. `not(a < b) or not (a < (b + a))`
19. `((a == b) and (a * a < b * b)) or ((b < a) and (2 * a < b))`
20. `((a == b) or not (b < a)) and ((a < b) or (b == a + 1))`

In Exercises 21 through 44, determine whether the condition evaluates to True or False.

21. `9W <> "9w"`
22. `'Harry' > 'Mine'`
23. `''Ab' == 'aB'`
24. `'D' >= '//'`
25. `'a' == 'A'`
26. `'1' < 'one'`
27. `("Duck" < "pig") and ("pig" < "big")`
28. `"Duck" < "Duck" + "Duck"`
29. `not(('B' == 'b') or ("Big" < "big"))`
30. `"th" in "Python"`
31. `"ty" in "Python"`
32. `7 < 34 and (not ("7" > "34" or "7" == "34"))`
33. `isinstance(32, float)`
34. `isinstance(32., int)`
35. `isinstance(32., float)`
36. `isinstance(32, int)`
37. `"colonel".startswith('k')`
38. `"knight".startswith('n')`
39. `potato.endswith("o",2,4)`
40. `"flute".endswith('t')`
41. `True or False`
42. `True and False`
43. `not True`
44. `not False`

In Exercises 45 through 54, determine whether or not the two conditions are equivalent—that is, whether they will both evaluate to True or both evaluate to False for any values of the variables appearing in them.

45. `a <= b; (a < b) or (a == b)`
46. `not(a < b); a > b`
47. `(a == b) or (a < b); a != b`
48. `not((a == b) or (a == c)); (a != b) and (a != c)`
49. `not((a == b) and (a == c)); (a != b) or (a != c)`
50. `(a < b) and ((a > d) or (a > e));`
    `((a < b) and (a > d)) or ((a < b) and (a > e))`

51. `(a <= b) and (a <= c); not((a > b) or (a > c))`

52. `not(a >= b); (a <= b) and not(a == b)`

53. `ch in "abcdefghijklmnopqrstuvwxyz"; 97 <= ord(ch) <= 122`

54. `str1.upper() == str1; str1.isupper()` (Assume *str1* has at least 1 alphabetic character.)

In Exercises 55 through 60, write a condition equivalent to the negation of the given condition. (For example, a != b is equivalent to the negation of a == b.)

55. `a > b`

56. `(a == b) or (a == d)`

57. `(a < b) and (c != d)`

58. `not((a == b) or (a > b))`

59. `a <= b`

60. `(a != "") and (a < b) and (len(a) < 5)`

In Exercises 61 through 68, simplify the expression. (In Exercises 63 through 68, assume that the variable has an integer value.)

61. `(ans == 'Y') or (ans == 'y') or (ans == "Yes") or (ans == "yes")`

62. `(name == "Athos") or (name == "Porthos") or (name == "Aramis")`

63. `(year == 2010) or (year == 2011) or (year == 2012) or (year == 2013)`

64. `(n == 1) or (n == 2) or (n == 3) or (n == 4) or (n == 5) or (n == 6)`

65. `(n >= 3) and (n < 9)`

66. `(n <= 22) and (n > 1)`

67. `(n <= 10) and (n > -20)`

68. `(n <= 200) and (n >= 100)`

In Exercises 69 through 84, determine whether True or False is displayed.

69.
```
str1 = "target"
print(str1.startswith('t') and str1.endswith('t'))
```

70.
```
print("colonel".startswith('k'))
```

71.
```
str1 = "target"
print(str1.startswith('t') or str1.endswith('t'))
```

72.
```
str1 = "target"
str2 = "get"
print(str1.startswith(str2,3))
```

73.
```
str1 = "Teapot"
str2 = "Tea"
print(str1.startswith(str2))
```

74.
```
str1 = "Teapot"
print(str1.startswith(str1[0:4]))
```

75.
```
str1 = "tattarrattat"
print(str1.endswith(str1[::-1]))
```

76.
```
str1 = "spam and eggs"
print(str1.endswith(str1[10:len(str1)]))
```

77.
```
str1 = "spam and eggs"
print(str1.startswith(str1[:len(str1) - 1]))
```

78.
```
num = "1234.56"
print(isinstance(num, float))
```

79.
```
print(isinstance(25.0, int))
```

80.
```
num = object()
print(isinstance(num, int))
```

81.
```
char = chr(80)
print(isinstance(char, int))
```

82.
```
print(isinstance('34', str))
```

83.
```
str1 = "-123"
print(str1.isdigit())
```

84.
```
print("seven".isdigit())
```

85. Rewrite the following statement using the chr function instead of escape sequences.
```
print("He said \"How ya doin?\" to me.")
```

Solutions to Practice Problems 3.1

1. False. The first string has six characters, whereas the second string has five. Two strings must be 100% identical to be equal.
2. When 27 and 9 are compared as strings, their first characters, 2 and 9, determine their order. Since 2 precedes 9 in the ASCII table, "27" < "9".
3.

cond1	cond2	cond1 and cond2	cond1 or cond2	not cond2
True	True	True	True	False
True	False	False	True	True
False	True	False	True	False
False	False	False	False	True

4. Yes. Apply the list function to the tuple in order to create a list containing the items in *monarch*. Then, apply the sort method to the list and use the tuple function to convert the list back into a tuple.
5. False. The space is not an alphabetical character.
6. The symbol = is used for assignment, whereas the symbol == is used for comparison.

3.2 Decision Structures


VideoNote
Decision Structures

Decision structures (also known as *branching structures*) allow a program to decide on a course of action based on whether a certain condition is true or false.

if-else Statements

An **if-else statement** is a statement of the form

```
if condition:
    indented block of statements
else:
    indented block of statements
```

causes the program to execute the first block of statements when the condition is true and to execute the second block of statements when the condition is false. Each indented block consists of one or more Python statements. The reserved words if and else must be written entirely in lowercase letters and each line in the blocks of statements should be indented

the same distance to the right. That is, they should be lined up vertically in columns. This physical indentation tells the interpreter and the human reader where the block starts and stops. (We will always indent blocks of statements by four spaces.)

Example 1 **Find the Larger Value** The following program finds the larger of two numbers input by the user. The condition is

```
num1 > num2
```

and each block consists of a single assignment statement. With the inputs 3 and 7 for *num1* and *num2*, the condition is false, and so the second block is executed. Figure 3.1 shows the flowchart for the **if-else** part of the program.

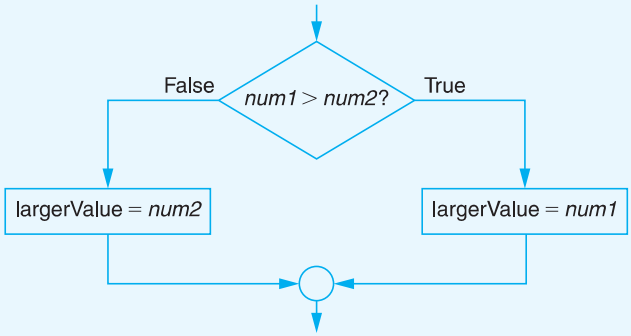

FIGURE 3.1 **Flowchart for the if-else statement in Example 1.**

```
## Determine the larger of two numbers.
# Obtain the two numbers from the user.
num1 = eval(input("Enter the first number: "))
num2 = eval(input("Enter the second number: "))
# Determine and display the larger value.
if num1 > num2:
    largerValue = num1  # execute this statement if the condition is true
else:
    largerValue = num2  # execute this statement if the condition is false
print("The larger value is", str(largerValue) + ".")
```

[Run]

```
Enter the first number: 3
Enter the second number: 7
The larger value is 7.
```

Example 2 **Volume of a Ten-Gallon Hat** The **if-else** statement in the following program has relational operators in its condition.

```
## A quiz.
# Obtain answer to question.
answer = eval(input("How many gallons does a ten-gallon hat hold? "))
# Evaluate answer.
if (0.5 <= answer <= 1):
    print("Good, ", end="")
```

```
else:
    print("No, ", end="")
print("it holds about 3/4 of a gallon.")
```

[Run]

```
How many gallons does a ten-gallon hat hold? 10
No, it holds about 3/4 of a gallon.
```

if Statements

The **else** part of an **if-else** statement can be omitted. If so, when the condition is false execution continues with the line after the **if** statement block. This type of **if** statement appears twice in the next example.

Example 3 **Find the Largest Value** The following program contains two **if** statements. Figure 3.2 shows the flowchart for the second part of the program. The value of *max* (largest value) is initially set to the first number input and is updated by the **if** statements when necessary.

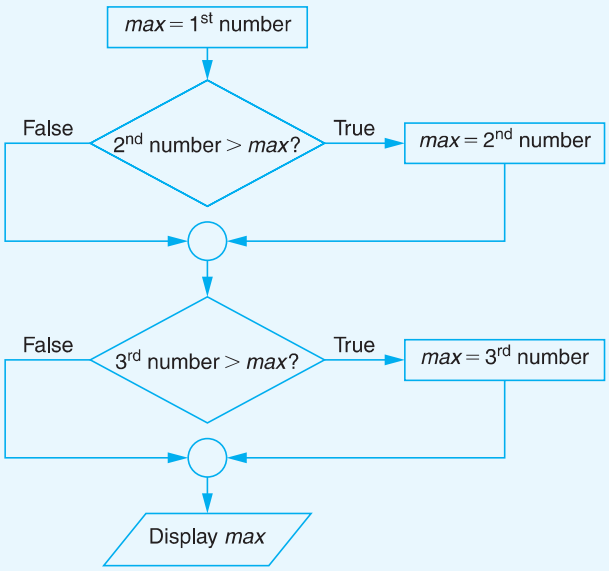

FIGURE 3.2 Flowchart for Example 3.

```
## Find the largest of three numbers.
# Input the three numbers.
firstNumber = eval(input("Enter first number: "))
secondNumber = eval(input("Enter second number: "))
thirdNumber = eval(input("Enter third number: "))
# Determine and display the largest value.
max = firstNumber
if secondNumber > max:
    max = secondNumber
if thirdNumber > max:
    max = thirdNumber
print("The largest number is", str(max) + ".")
```

[Run]

```
Enter first number: 3
Enter second number: 7
Enter third number: 4
The largest number is 7.
```

Nested *if-else* Statements

The indented blocks of **if-else** and **if** statements can contain other **if-else** and **if** statements. In this situation the statements are said to be **nested**. Examples 4 and 5 contain nested **if-else** statements.

Example 4 Interpret Beacon The color of the beacon light atop Boston's old John Hancock building forecasts the weather according to the following rhyme:

Steady blue, clear view.
Flashing blue, clouds due.
Steady red, rain ahead.
Flashing red, snow instead.

The following program requests a color (Blue or Red) and a mode (Steady or Flashing) as input and then displays the weather forecast. Both courses of action associated with the main **if-else** statement consist of **if-else** statements.

```
## Interpret weather beacon.
# Obtain color and mode.
color = input("Enter a color (BLUE or RED): ")
mode = input("Enter a mode (STEADY or FLASHING): ")
color = color.upper()
mode = mode.upper()
# Analyze responses and display weather forecast.
result = ""
if color == "BLUE":
    if mode == "STEADY":
        result = "Clear View."
    else:  # mode is FLASHING
        result = "Clouds Due."
else:  # color is RED
    if mode == "STEADY":
        result = "Rain Ahead."
    else:  # mode is FLASHING
        result = "Snow Ahead."
print("The weather forecast is", result)
```

[Run]

```
Enter the color (BLUE or RED): RED
Enter the mode (STEADY or FLASHING): STEADY
The weather forecast is Rain Ahead.
```

Example 5 **Evaluate Profit** The following program requests the costs and revenue for a company and displays the message "Break even" if the costs and revenue are equal; otherwise, it displays the profit or loss. The indented block following the **else** header is another **if-else** statement.

```
## Evaluate profit.
# Obtain input from user.
costs = eval(input("Enter total costs: "))
revenue = eval(input("Enter total revenue: "))
# Determine and display profit or loss.
if costs == revenue:
    result = "Break even."
else:
    if costs < revenue:
        profit = revenue - costs
        result = "Profit is ${0:,.2f}.".format(profit)
    else:
        loss = costs - revenue
        result = "Loss is ${0:,.2f}.".format(loss)
print(result)
```

[Run]

```
Enter total costs: 9500
Enter total revenue: 8000
Loss is $1,500.00.
```

The *elif* Clause

An extension of the **if-else** statement allows for more than two possible alternatives with the inclusion of **elif** clauses. (**elif** is an abbreviation for "else if.") A typical compound statement containing **elif** clauses is as follows:

```
if condition1:
    indented block of statements to execute if condition1 is true
elif condition2:
    indented block of statements to execute if condition2 is true
    AND condition1 is not true
elif condition3:
    indented block of statements to execute if condition3 is true
    AND both previous conditions are not true
else:
    indented block of statements to execute if none of the above
    conditions are true
```

Python searches for the first true condition and carries out its associated block of statements. If none of the conditions are true, then **else**'s block of statements is carried out. Execution then continues with the statement following the **if-elif-else** statement. In general, an **if-elif-else** statement can contain any number of **elif** clauses. As before, the **else** clause is optional.

Example 6 **Find the Larger Value** The following program modifies Example 1 so that the program reports if the two numbers are equal.

```
## Determine the larger of two numbers.
# Obtain the two numbers from the user.
num1 = eval(input("Enter the first number: "))
num2 = eval(input("Enter the second number: "))
# Determine and display the larger value.
if num1 > num2:
    print("The larger value is", str(num1) + ".")
elif num2 > num1:
    print("The larger value is", str(num2) + ".")
else:
    print("The two values are equal.")
```

[Run]

```
Enter the first number: 7
Enter the second number: 7
The two values are equal.
```

The **if-elif** statement in Example 7 allows us to calculate values that are not determined by a simple formula.

Example 7 **FICA Tax** The Social Security or FICA tax has two components—the Social Security benefits tax, which in 2014 was 6.2% of the first $117,000 of earnings for the year, and the Medicare tax, which was 1.45% of earnings plus .9% of earnings above $200,000 (for unmarried employees). The following program calculates a single employee's FICA tax withheld for the current pay period.

```
## Calculate FICA tax for a single employee.
# Obtain earnings.
str1 = "Enter total earnings for this year prior to current pay period: "
ytdEarnings = eval(input(str1))  # year-to-date earnings
curEarnings = eval(input("Enter earnings for the current pay period: "))
totalEarnings = ytdEarnings + curEarnings
# Calculate the Social Security Benefits tax.
socialSecurityBenTax = 0
if totalEarnings <= 117000:
    socialSecurityBenTax = 0.062 * curEarnings
elif ytdEarnings < 117000:
    socialSecurityBenTax = 0.062 * (117000 - ytdEarnings)
# Calculate and display the FICA tax.
medicareTax = 0.0145 * curEarnings
if ytdEarnings >= 200000:
    medicareTax += 0.009 * curEarnings
elif totalEarnings > 200000:
    medicareTax += 0.009 * (totalEarnings - 200000)
ficaTax = socialSecurityBenTax + medicareTax
print("FICA tax for the current pay period: ${0:0,.2f}".format(ficaTax))
```

[Run]

```
Enter total earnings for this year prior to current pay period: 12345.67
Enter earnings for the current pay period: 543.21
FICA tax for current pay period: $41.56
```

The following example illustrates the fact that when a decision construct contains **elif** clauses, Python executes the block of statements corresponding to the first condition that is satisfied and ignores all subsequent **elif** clauses—even if they also satisfy the condition.

Example 8 **Graduation Honors** The program in Fig. 3.3 assumes that the user will graduate and determines if they will graduate with honors.

```
## Bestow graduation honors.
# Request grade point average.
gpa = eval(input("Enter your gpa: "))
# Determine if honors are warranted.
if gpa >= 3.9:
    honors = " summa cum laude."
elif gpa >= 3.6:
    honors = " magna cum laude."
elif gpa >= 3.3:
    honors = " cum laude."
else:
    honors = "."
# Display conclusion.
print("You graduated" + honors)
```

[Run]

```
Enter your gpa: 3.7
You graduated magna cum laude.
```

[Run]

```
Enter your gpa: 3
You graduated.
```

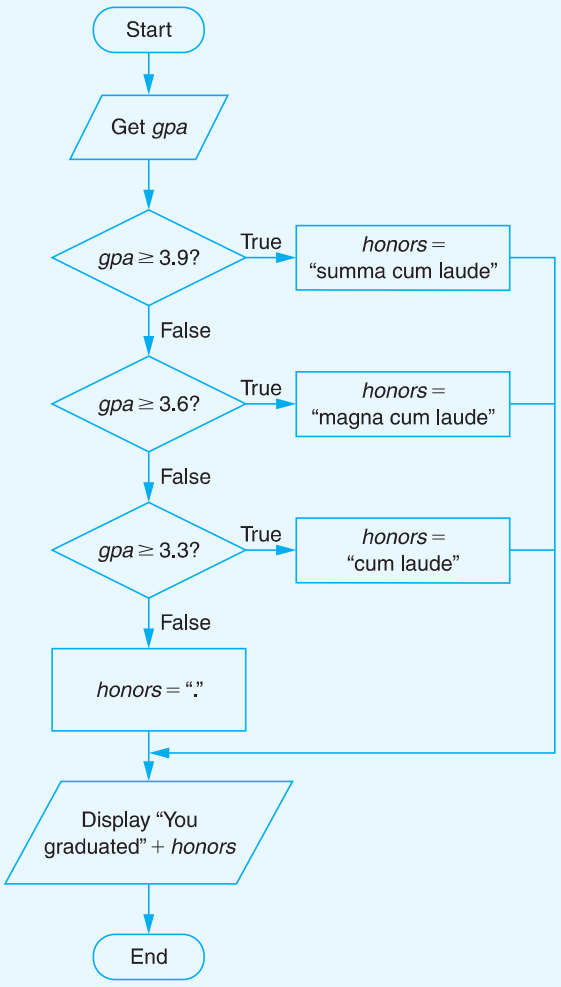

FIGURE 3.3 **Program and Flowchart for Example 8.**

Input Validation with *if-elif-else* Statements

Suppose a program asks the user to input a number, and then uses the number in a calculation. If the user does not enter a number or enters an inappropriate number, the program will crash. The Boolean-valued method **isdigit** can be used to prevent this from happening.

Example 9 Input Validation The following program uses the method isdigit to guard against improper input.

```
## Request two numbers and find their sum. Validate the input.
num1 = input("Enter first number: ")
num2 = input("Enter second number: ")
# Display sum if entries are valid. Otherwise, inform
# the user where invalid entries were made.
if num1.isdigit() and num2.isdigit():
    print("The sum is", str(eval(num1) + eval(num2)) + ".")
elif not num1.isdigit():
    if not num2.isdigit():
        print("Neither entry was a proper number.")
    else:
        print("The first entry was not a proper number.")
else:
    print("The second entry was not a proper number.")
```

[Run]

```
Enter first number: 5
Enter second number: six
The second entry was not a proper number.
```

True and False

Every object has a truth value associated with it and therefore can be used as a condition. When numbers are used as conditions, 0 evaluates to False and all other numbers evaluate to True. Of course, the objects True and False evaluate to True and False, respectively. A string, list, or tuple used as a condition evaluates to False if it is empty, and otherwise evaluates to True.

Example 10 True or False The following program illustrates the truth values of objects.

```
## Illustrate Boolean values.
if 7:
    print("A nonzero number is true.")
else:
    print("The number zero is false.")
if []:
    print("A nonempty list is true.")
else:
    print("An empty list is false.")
if ["spam"]:
    print("A nonempty list is true.")
else:
    print("The empty list is false.")
```

[Run]

```
A nonzero number is true.
An empty list is false.
A nonempty list is true.
```

Comments

1. A line of the form `if boolExp == True:` should be shortened to `if boolExp:`. Similarly, a line of the form `if boolExp == False:` should be shortened to `if not boolExp:`.
2. `if` statements can be used to guarantee that a number input by the user is in the proper range. For instance, when the user is asked to input an exam grade, a line such as

   ```
   if (0 <= grade <= 100):
   ```

   can be used to guarantee that the number entered is between 0 and 100.
3. The words `if`, `else`, and `elif` are reserved words and therefore are colorized orange by IDLE.
4. The use of indentation to mark blocks of code helps make Python code more readable. IDLE helps with indentation by automatically indenting code when required. For instance, when the *Enter* (or *return*) key is pressed after the colon is typed at the end of an `if`, `elif`, or `else` header, IDLE automatically indents the next line of code.
5. Statements consisting of a header followed by an indented block of code are called **compound statements**. Two other compound statements, the `while` statement and the `for` statement, are discussed in the next two sections.
6. The last six lines of Example 6 could have been written without `elif` as follows:

   ```
   if num1 > num2:
       print("The larger value is", str(num1) + ".")
   if num2 > num1:
       print("The larger value is", str(num2) + ".")
   if num2 = num1:
       print("The two values are equal.")
   ```

   However, `elif` should always be used when the test conditions are mutually exclusive. In the code above, all three `if` statements are executed even if the first one is true.

Practice Problems 3.2

1. Suppose the user is asked to input a number for which the square root is to be taken. Complete the `if` statement so that the lines of code that follow will display either the message "Number can't be negative." or will display the square root of the number.

   ```
   # Check reasonableness of input.
   number = eval(input("Enter a non-negative number: "))
   if
   ```

2. Improve the following code:

   ```
   if a < b:
       if c < 5:
           print("hello")
   ```

3. Improve the following code:

   ```
   if (name == "John") or (name == "George") or \
       (name == "Paul") or (name == "Ringo"):
       flag = True
   else:
       flag = False
   ```

4. Rewrite Example 5 using `elif`.

EXERCISES 3.2

In Exercises 1 through 14, determine the output displayed.

1.
```
num = 4
if num <= 9:
    print("Less than ten.")
elif num == 4:
    print("Equal to four.")
```

2.
```
gpa = 3.49
result = ""
if gpa >= 3.5:
    result = "Honors"
print(result + "Student")
```

3.
```
print('a+b' < 'b+c')
```

4.
```
print('a*b' < 'b*c')
```

5.
```
a = 5
b = 7
sentence = ""
if ((5 * a) - 2*b + 4) <= (3*b-1):
    sentence = "Remember,"
print(sentence + "tomorrow is New Year's Day.")
```

6.
```
change = 356
if change >= 100:
    print("Your change contains", change // 100, "dollars.")
else:
    print("Your change contains no dollars.")
```

7.
```
a = 2
b = 3
c = 7
if (a * b) < c:
    b = a
else:
    c = a+b+c
print(a, b, c)
```

8.
```
length = eval(input("Enter length of cloth in yards: "))
if length < 1:
    cost = 3.00    # cost in dollars
else:
    cost = 3.00 + ((length - 1) * 2.50)
result = "Cost of cloth is ${0:0.2f}.".format(cost)
print(result)
```
(Assume the response is 6.)

9.
```
letter = input("Enter A, B, or C: ")
letter = letter.upper()
if letter == "A":
    print("A, my name is Alice.")
elif letter == "B":
    print("To be, or not to be.")
elif letter == "C":
    print("Oh, say, can you see.")
else:
    print("You did not enter a valid letter.")
```
(Assume the response is *B*.)

10.
```
isvowel = False
letter = input("Enter a letter: ")
letter = letter.upper()
if (letter in "AEIOU"):
    isvowel = True
if isvowel:
    print(letter, "is a vowel.")
elif (not(65 <= ord(letter) <= 90)):
    print("You did not enter a letter.")
else:
    print(letter, "is not a vowel.")
```
(Assume the response is *a*.)

11.
```
a = 5
if (a > 2) and ((a == 3) or (a < 7)):
    print("Hi")
```

12.
```
number = 5
if number < 0:
    print("negative")
else:
    if number == 0:
        print("zero")
    else:
        print("positive")
```

13.
```
if "spam":
    print("A nonempty string is true.")
else:
    print("A nonempty string is false.")
```

14.
```
if "":
    print("An empty string is true.")
else:
    print("An empty string is false.")
```

In Exercises 15 through 18, identify the errors, state the type of each error (syntax, runtime, or logic), and correct the block of code.

15.
```
n = eval(input("Enter a number: "))
if 'n'%2 = 0:
    print("The number is an even number.")
else:
    print("The number is an odd number.")
```

16.
```
number = 6
if number > 5 and < 9:
    print("Yes")
else:
    print("No")
```

17.
```
major = "Computer Science"
if major == "Business" Or "Computer Science":
    print("Yes" )
```

18.
```
if a not b:
    print("Both are unequal")
else:
    print(“Both are equal”)
```

In Exercises 19 through 24, simplify the code.

19.
```
if (a*3%3):
    a = (a*a)/(a+a)
else:
    a=5
```

20.
```
if (a == 7):
    print("seven")
elif (a != 7):
    print("eleven")
```

21.
```
if (j == 7):
    b = 1
else:
    if (j != 7):
        b = 2
```

22.
```
if state == "CA":
    if city == "LA" or city == "SD":
        print("Large city!")
```

23.
```
answer = input("Is the Indian Ocean bigger than the Pacific Ocean?")
if (answer[0]=="Y"):
    answer="YES"
elif (answer[0]=="y"):
    answer="YES"
if(answer=="YES"):
    print("Incorrect")
elif(answer=="NO"):
    print("Correct")
```

24.
```
feet = eval(input("How tall (in feet) is the Statue of Liberty? "))
if (feet <= 141):
    print("Nope")
if (feet > 141):
    if (feet < 161):
        print("Good")
    else:
        print("Nope")
print("The statue is 151 feet tall from base to torch.")
```

25. **Restaurant Tip** Write a program to determine how much to tip the server in a restaurant. The tip should be 15% of the check, with a minimum of $2. See Fig. 3.4.

```
Enter amount of bill: 25.98
Tip is $3.90
```

FIGURE 3.4 Possible outcome of Exercise 25.

```
Enter number of bagels: 12
Cost is $7.20.
```

FIGURE 3.5 Possible outcome of Exercise 26.

26. **Cost of Bagels** A bagel shop charges 75 cents per bagel for orders of less than a half-dozen bagels and 60 cents per bagel for orders of a half-dozen or more. Write a program that requests the number of bagels ordered and displays the total cost. See Fig. 3.5.

27. **Cost of Widgets** A store sells widgets at 25 cents each for small orders or at 20 cents each for orders of 100 or more. Write a program that requests the number of widgets ordered and displays the total cost. See Fig. 3.6.

```
Enter number of widgets: 200
Cost is $40.00
```

FIGURE 3.6 Possible outcome of Exercise 27.

```
Enter number of copies: 125
Cost is $5.75.
```

FIGURE 3.7 Possible outcome of Exercise 28.

28. Cost of Copies A copy center charges 5 cents per copy for the first 100 copies and 3 cents per copy for each additional copy. Write a program that requests the number of copies as input and displays the total cost. See Fig. 3.7.

29. Quiz Write a quiz program to ask "Who was the first Ronald McDonald?" The program should display "You are correct." if the answer is "Willard Scott" and "Nice try." for any other answer. See Fig. 3.8.

```
Who was the first Ronald McDonald? Willard Scott
You are correct.
```

FIGURE 3.8 Possible outcome of Exercise 29.

30. Overtime Pay Federal law requires that hourly employees be paid "time-and-a-half" for work in excess of 40 hours in a week. For example, if a person's hourly wage is $12 and he or she works 60 hours in a week, the person's gross pay should be

$$(40 * 12) + (1.5 * 12 * (60 - 40)) = \$840.$$

Write a program that requests the number of hours a person works in a given week and the person's hourly wage as input, and then displays the person's gross pay. See Fig. 3.9.

31. Compute an Average Write a program that requests three scores as input and displays the average of the two highest scores. See Fig. 3.10.

```
Enter hourly wage: 12.50
Enter number of hours worked: 47
Gross pay for week is $631.25.
```

FIGURE 3.9 Possible outcome of Exercise 30.

```
Enter first score: 85
Enter second score: 93
Enter third score: 91
Average of two highest
scores is 92.00
```

FIGURE 3.10 Possible outcome of Exercise 31.

32. Pig Latin Write a program that requests a word (in lowercase letters) as input and translates the word into Pig Latin. See Fig. 3.11. The rules for translating a word into Pig Latin are as follows:

(a) If the word begins with a group of consonants, move them to the end of the word and add *ay*. For instance, *chip* becomes *ipchay*.

(b) If the word begins with a vowel, add *way* to the end of the word. For instance, *else* becomes *elseway*.

```
Enter word to translate: chip
The word in Pig Latin is ipchay.
```

FIGURE 3.11 Possible outcome of Exercise 32.

```
Enter weight in pounds: 6
Enter payment in dollars: 20
Your change is $5.00.
```

FIGURE 3.12 Possible outcome of Exercise 33.

33. Make Change A supermarket sells apples for $2.50 per pound. Write a cashier's program that requests the number of pounds and the amount of cash tendered as input and displays the change from the transaction. If the cash is not enough, the message "You owe $x.xx more." should be displayed, where $x.xx is the difference between the total cost and the cash. See Fig. 3.12 on the previous page.

34. Savings Account Write a program to process a savings-account withdrawal. The program should request the current balance and the amount of the withdrawal as input and then display the new balance. If the withdrawal is greater than the original balance, the program should display "Withdrawal denied." If the new balance is less than $150, the message "Balance below $150" should also be displayed. See Fig. 3.13.

```
Enter current balance: 200
Enter amount of withdrawal: 25
The new balance is $175.00.
```

FIGURE 3.13 Possible outcome of Exercise 34.

```
Enter a single uppercase letter: TEE
You did not comply with the request.
```

FIGURE 3.14 Possible outcome of Exercise 35.

35. Input Validation Write a program that asks the user to enter a single uppercase letter and then informs the user if they didn't comply with the request. See Fig. 3.14.

36. Year The current calendar, called the Gregorian calendar, was introduced in 1582. Every year divisible by four was created to be a leap year, with the exception of the years ending in 00 (that is, those divisible by 100) and not divisible by 400. For instance, the years 1600 and 2000 are leap years, but 1700, 1800, and 1900 are not. Write a program that requests a year as input and states whether it is a leap year. See Fig. 3.15.

```
Enter a year: 2016
2016 is a leap year.
```

FIGURE 3.15 Possible outcome of Exercise 36.

```
Enter a military time (0000 to
2359): 1532
The regular time is 3:32 pm.
```

FIGURE 3.16 Possible outcome of Exercise 37.

37. Military Time In military time, hours are numbered from 00 to 23. Under this system, midnight is 00, 1 a.m. is 01, 1 p.m. is 13, and so on. Time in hours and minutes is given as a four-digit string with minutes following hours and given by two digits ranging from 00 to 59. For instance, military time 0022 corresponds to 12:22 a.m. regular time, and military time 1200 corresponds to noon regular time. Write a program that converts from military time to regular time. See Fig. 3.16.

38. Railroad Properties One of the four railroad properties in Monopoly is not an actual railroad. Write a program that displays the names of the four properties and asks the user to identify the property that is not a railroad. The user should be informed if the selection is correct or not. See Fig. 3.17.

```
The four railroad properties
are Reading, Pennsylvania,
B & O, and Short Line.
Which is not a railroad? Short Line
Correct.
Short Line is a bus company.
```

FIGURE 3.17 Possible outcome of Exercise 38.

39. Interest Rates Savings accounts state an interest rate and a compounding period. If the amount deposited is P, the stated interest rate is r, and interest is compounded m times per year, then the balance in the account after one year is $P \cdot \left(1 + \frac{r}{m}\right)^m$. For instance, if \$1,000 is deposited at 3% interest compounded quarterly (that is, four times per year), then the balance after one year is

$$1{,}000 \cdot \left(1 + \frac{.03}{4}\right)^4 = 1{,}000 \cdot 1.0075^4 = \$1{,}030.34.$$

Interest rates with different compounding periods cannot be compared directly. The concept of APY (annual percentage yield) must be used to make the comparison. The APY for a stated interest rate r compounded m times per year is defined by

$$\text{APY} = \left(1 + \frac{r}{m}\right)^m - 1.$$

(The APY is the simple interest rate that yields the same amount of interest after one year as the compounded annual rate of interest.) Write a program to compare interest rates offered by two different banks and determine the most favorable interest rate. See Fig. 3.18.

```
Enter annual rate of interest for Bank 1: 2.7
Enter number of compounding periods for Bank 1: 2
Enter annual rate of interest for Bank 2: 2.69
Enter number of compounding periods for Bank 2: 52
APY for Bank 1 is 2.718%.
APY for Bank 2 is 2.726%.
Bank 2 is the better bank.
```

FIGURE 3.18 Possible outcome of Exercise 39.

40. Graduation Honors Rewrite the program in Example 8 without **elif** clauses. That is, the task should be carried out with a sequence of simple **if** statements.

41. Graduation Honors Rewrite the program in Example 8 so that the GPA is validated to be between 2 and 4 before the **if-elif-else** statement is executed.

42. Second-Suit-Half-Off Sale A men's clothing store advertises that if you buy a suit, you can get a second suit at half-off. What they mean is that if you buy two suits, then the price of the lower-cost suit is reduced by 50%. Write a program that accepts the two costs as input and then calculates the total cost after halving the cost of the lowest price suit. See Fig. 3.19.

```
Enter cost of first suit: 378.50
Enter cost of second suit: 495.99
Cost of the two suits is $685.24
```

FIGURE 3.19 Possible outcome of Exercise 42.

```
Enter your taxable income: 60000
Your tax is $1,500.
```

FIGURE 3.20 Possible outcome of Exercise 43.

43. Income Tax The flowchart in Fig. 3.21 on the next page calculates a person's state income tax. Write a program corresponding to the flowchart. See Fig. 3.20.

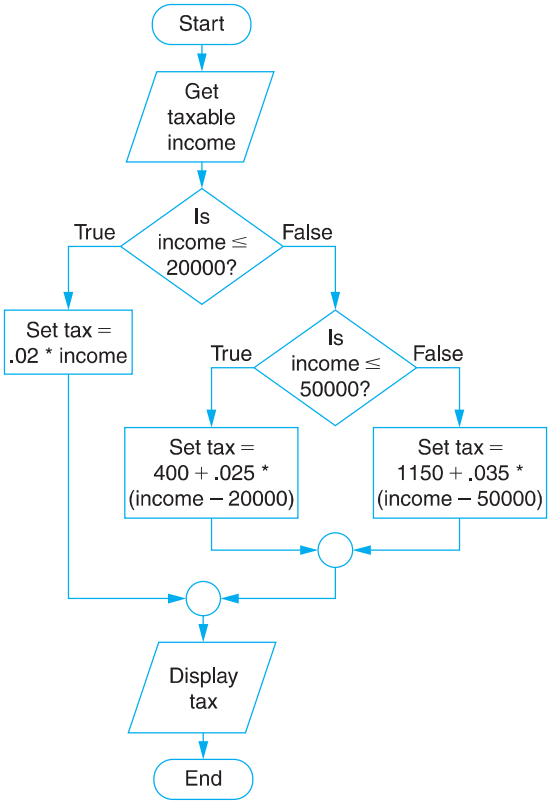

FIGURE 3.21 Flowchart for Exercise 43.

Solutions to Practice Problems 3.2

1.
```
# Check reasonableness of input.
number = eval(input("Enter a non-negative number: "))
if number >= 0:
    print("The square root of the number is", str(number ** .5)+ ".")
else:
    print("Number can't be negative.")
```

2. The word *hello* will be displayed when (a < b) is true and (c < 5) is also true. That is, it will be displayed when both of these two conditions are true. The clearest way to write the code is

```
if (a < b) and (c < 5):
    print("hello")
```

3.
```
flag = name in ["John", "George", "Paul", "Ringo"]
```

4.
```
## Evaluate profit.
# Obtain input from user.
costs = eval(input("Enter total costs: "))
revenue = eval(input("Enter total revenue: "))
# Determine and display profit or loss.
if costs == revenue:
    result = "Break even."
elif costs < revenue:
    profit = revenue - costs
    result = "Profit is ${0:,.2f}.".format(profit)
else:
```

```
    loss = costs - revenue
    result = "Loss is ${0:,.2f}.".format(loss)
print(result)
```

3.3 The *while* Loop

A **loop**, one of the most important structures in programming, is a part of a program that can execute a block of code repeatedly.

The *while* Loop


VideoNote
The *while* Loop

The **while loop** repeatedly executes an indented block of statements as long as a certain condition is met. A while loop has the form

```
while condition:
    indented block of statements
```

The line beginning with while is called the **header** of the loop, the condition in the header is called the **continuation condition** of the loop, the indented block of code is called the **body** of the loop, and each execution of the body is called a **pass** through the loop. The continuation condition is a Boolean expression that evaluates to either True or False. Each line in the block of statements should be indented the same distance to the right. This physical indentation of the block tells the interpreter where the block starts and stops.

When Python encounters a while loop, it first checks the truth value of the continuation condition. If the condition evaluates to False, Python skips over the body of the loop and continues with the line (if any) after the loop. If the continuation condition evaluates to True, the body of the loop is executed. After each pass through the loop, Python rechecks the condition and proceeds accordingly. That is, the body will be continually executed until the continuation condition evaluates to False.

Example 1 **Numbers** The program in Fig. 3.22, in which the continuation condition is `num <= 5`, displays the numbers from 1 through 5. After the loop terminates, the value of *num* will be 6.

```
## Display the numbers from 1 to 5.
num = 1
while num <= 5:
    print(num)
    num += 1  # Increase the value of num by 1.
```

[Run]

```
1
2
3
4
5
```

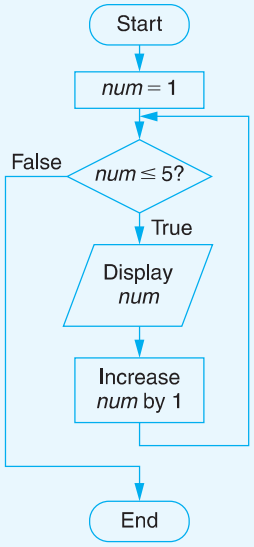

FIGURE 3.22 **Program and Flowchart for Example 1.**

A **while** loop can be used to ensure that a proper response is received from a request for input. This process is called **input validation**.

Example 2 **Movie Quotations** The following program requires the user to enter a number from 1 through 3. The loop repeats the request until the user gives an acceptable response.

```
## Movie Quotations
print("This program displays a famous movie quotation.")
responses = ('1', '2', '3')
response = '0'
while response not in responses:
    response = input("Enter 1, 2, or 3: ")
    if response == '1':
        print("Plastics.")
    elif response == '2':
        print("Rosebud.")
    elif response == '3':
        print("That's all folks.")
```

[Run]

```
This program displays a famous movie quotation.
Enter 1, 2, or 3: one
Enter 1, 2, or 3: 5
Enter 1, 2, or 3: 2
Rosebud.
```

Example 3 **Numbers** The following program finds the minimum, maximum, and average of a sequence of nonnegative numbers entered by the user. The user is told to enter the number −1 to indicate the end of data entry. Since the first request for input appears before the loop is entered, there is the possibility that the entire loop will be skipped. The values of *min* and *max* are initially set to the first number input and are updated during each pass through the loop.

```
## Find the minimum, maximum, and average of a sequence of numbers.
count = 0  # number of nonnegative numbers input
total = 0  # sum of the nonnegative numbers input
# Obtain numbers and determine count, min, and max.
print("(Enter -1 to terminate entering numbers.)")
num = eval(input("Enter a nonnegative number: "))
min = num
max = num
while num != -1:
    count += 1
    total += num
    if num < min:
        min = num
    if num > max:
        max = num
    num = eval(input("Enter a nonnegative number: "))
```

```
# Display results.
if count > 0:
    print("Minimum:", min)
    print("Maximum:", max)
    print("Average:", total / count)
else:
    print("No nonnegative numbers were entered.")
```

[Run]

```
(Enter −1 to terminate entering numbers.)
Enter a nonnegative number: 3
Enter a nonnegative number: 7
Enter a nonnegative number: 2
Enter a nonnegative number: -1
Minimum: 2
Maximum: 7
Average: 4.0
```

In Example 3, the variable *count* is called a **counter variable**, the variable *total* is called an **accumulator variable**, the number −1 is called a **sentinel value**, and the loop is referred to as having **sentinel-controlled repetition**.

Example 4 Numbers The following program performs the same tasks as the program in Example 3. However, it first stores the numbers in a list, and then uses list methods and functions to determine the requested values.

```
## Find the minimum, maximum, and average of a sequence of numbers.
# Obtain list of numbers.
list1 = []
print("(Enter -1 to terminate entering numbers.)")
num = eval(input("Enter a nonnegative number: "))
while num != -1:
    list1.append(num)
    num = eval(input("Enter a nonnegative number: "))
# Display results.
if len(list1) > 0:
    list1.sort()
    print("Minimum:", list1[0])
    print("Maximum:", list1[-1])
    print("Average:", sum(list1) / len(list1))
else:
    print("No nonnegative numbers were entered.")
```

Loops allow us to calculate useful quantities for which we might not know a simple formula.

Example 5 **Compound Interest** Suppose you deposit money into a savings account and let it accumulate at 4% interest compounded annually. The following program determines when you will be a millionaire.

```
## Calculate the number of years to become a millionaire.
numberOfYears = 0
balance = eval(input("Enter initial deposit: "))
while balance < 1000000:
    balance += .04 * balance
    numberOfYears += 1
print("In", numberOfYears, "years you will have a million dollars.")
```

[Run]

```
Enter initial deposit: 123456
In 54 years you will have a million dollars.
```

The *break* Statement

The **break** statement causes an exit from anywhere in the body of a loop. When the statement

```
break
```

is executed in the body of a **while** loop, the loop immediately terminates. **Break** statements usually appear in the bodies of **if** statements.

Example 6 **Numbers** The rewrite in Fig. 3.23 of the "Obtain list of numbers" code from Example 4 uses a **break** statement to avoid having two input statements. Many people find this rewrite easier to read.

```
# Obtain list of numbers.
list1 = []
print("(Enter -1 to terminate entering numbers.)")
while True:
    num = eval(input("Enter a nonnegative number: "))
    if num == -1:
        break   # Immediately terminate the loop.
    list1.append(num)
```

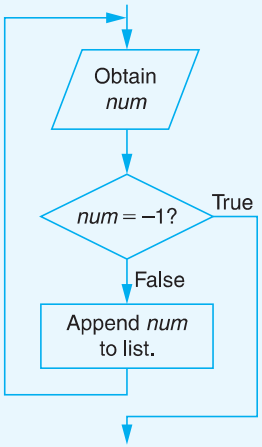

FIGURE 3.23 Code and Flowchart for Example 6.

The *continue* Statement

When the statement

```
continue
```

is executed in the body of a **while** loop, the current iteration of the loop terminates and execution returns to the loop's header. **Continue** statements usually appear inside **if** statements.

Example 7 **Integer Divisible by 11** The following program searches a list for the first **int** object that is divisible by 11. The variable *foundFlag* tells us if such an **int** has been found. (A **flag** is a Boolean-valued variable used to report whether a certain circumstance has occurred. The value of the flag is initially set to **False**, and then is changed to **True** if and when the circumstance occurs).

```
## Find first integer divisible by 11.
list1 = ["one", 23, 17.5, "two", 33, 22.1, 242, "three"]
i = 0
foundFlag = False
while i < len(list1):
    x = list1[i]
    i += 1
    if not isinstance(x, int):
        continue   # Skip to next item in list.
    if x % 11 == 0:
        foundFlag = True
        print(x, "is the first int that is divisible by 11.")
        break
if not foundFlag:
    print("There is no int in the list that is divisible by 11.")
```

[Run]

```
33 is the first int in the list that is divisible by 11.
```

Creating a Menu

Accessing menus is one of the fundamental tasks of interactive programs. The user makes choices until he or she decides to quit.

Example 8 **U.S. Facts** The following program uses a menu to obtain facts about the United States.

```
## Display facts about the United States.
print("Enter a number from the menu to obtain a fact")
print("about the United States or to exit the program.\n")
print("1. Capital")
print("2. National Bird")
print("3. National Flower")
print("4. Quit\n")
while True:
    num = int(input("Make a selection from the menu: "))
    if num == 1:
        print("Washington, DC is the capital of the United States.")
```

```
    elif num == 2:
        print("The American Bald Eagle is the national bird.")
    elif num == 3:
        print("The Rose is the national flower.")
    elif num == 4:
        break
```

[Run]

```
Enter a number from the menu to obtain a fact
about the United States or to exit the program.

1. Capital
2. National Bird
3. National Flower
4. Quit

Make a selection from the menu: 3
The Rose is the national flower.
Make a selection from the menu: 2
The American Bald Eagle is the national bird.
Make a selection from the menu: 4
```

Infinite Loops

Be careful to avoid **infinite loops**; that is, loops that never end.

Example 9 Infinite Loop The program in Fig. 3.24 contains an infinite loop because the condition `number >= 0` will always be true. ***Note:*** While an infinite loop is executing, you can terminate the program by clicking on *Close* in the IDLE *File* menu.

```
## Infinite loop.
print("(Enter -1 to terminate entering numbers.)")
number = 0
while number >= 0:
    number = eval(input("Enter a number to square: "))
    number = number * number
    print(number)
```

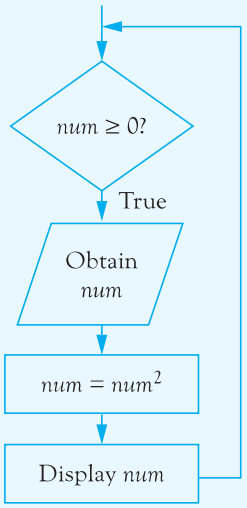

FIGURE 3.24 Program and Flowchart for Example 9.

Practice Problems 3.3

1. What is wrong with the following program?

```
initial_val = 10
while initial_val >0:
   print initial_val
   if initial_val == 5:
     break
   print initial_val
```

2. Change the following code segment so that the loop will execute at least once.

```
while answer.upper() != "SHAZAM":
    answer = input("Enter the password: "))
print("You may continue.")
```

3. How would you change the following code segment so that the word "Python" is displayed?

```
letters = ['P','y','t','h','o','n']
language = ""
i = 0
while letters: # This is the same as writing while letters != []
    language += letters [i]
    i = i+1
    letters = letters [i:]

print(language)
```

EXERCISES 3.3

In Exercises 1 through 8, determine the output displayed.

1.
```
num = 5
while True:
    num = 2 * num
    if num % 4 == 0:
        break
print(num)
```

2.
```
num = 3
while num < 15:
    num += 5
print(num)
```

3.
```
total = 0
num = 1
while True:
    total += num
    num += 1
    if num == 10:
        break
print(total)
```

4.
```
total = 0
num = 1
while num < 5:
    total += num
    num += 1
print(total)
```

5.
```
list1 = [2, 4, 6, 8]
total = 0
while list1: # same as while list1 != []:
    total += list1[0]
    list1 = list1[1:]
print(total)
```

6.
```
oceans = ["Atlantic", "Pacific", "Indian", "Arctic", "Antarctic"]
i = len(oceans) - 1
while i >= 0:
    if len(oceans[i]) < 7:
        del oceans[i]
    i = i - 1
print(", ".join(oceans))
```

7.
```
list1 = ['a', 'b', 'c', 'd']
i = 0
while True:
    print(list1[i]*i)
    i = i + 1
    if i == len(list1):
        break
```

8.
```
numTries = 0
year = 0
while (numTries < 7) and (year != 1964):
    numTries += 1
    year = int(input("Try #" + str(numTries) + ": In what year " +
                     "did the Beatles invade the U.S.? "))
    if year == 1964:
        print("\nYes. They performed on the Ed Sullivan show in 1964.")
        print("You answered correctly in " + str(numTries) + " tries.")
    elif year < 1964:
        print("Later than", year)
    else: # year > 1964
        print("Earlier than", year)
if (numTries == 7) and (year != 1964):
    print("\nYour 7 tries are up. The answer is 1964.")
```

(Assume that the responses are 1950, 1970, and 1964.)

In Exercises 9 through 12, identify the errors.

9.
```
q = 1
while q!= 0:
    q=q-2
    print(q)
```

10.
```
## Display the numbers from 1 through 5.
num = 0
while True
    num = 1
    print(num)
    num += 1
```

11.
```
## Display the elements of a list
list1 = ['H', 'e', 'l', 'l', 'o']
i = len(list1)
while i > 1:
    i - = 1
    print(list1[i])
```

12.
```
## Display the elements from a list.
list1 = ['a', 'b', 'c', 'd']
i = 0
while True:
    print(list1[i])
    if i = len(list1):
        break
    i = i + 1
```

In Exercises 13 and 14, write a simpler and clearer code that performs the same task as the given code.

13.
```
sum = int(input("Enter a number: "))
num = int(input("Enter a number: "))
sum = sum + num
num = int(input("Enter a number: "))
sum = sum + num
print(sum)
```

14.
```
L = [2, 4, 6, 8]
total = 0
while L != []:
    total += L[0]
    L = L[1:]
print(total)
```

15. Temperature Conversions Write a program that displays a Celsius-to-Fahrenheit conversion table. Entries in the table should range from 10 to 30 degrees Celsius in increments of 5 degrees. See Fig. 3.25. ***Note:*** The formula $f = \left(\frac{9}{5} \cdot c\right) + 32$ converts Celsius degrees to Fahrenheit degrees.

Celsius	Fahrenheit
10	50
15	59
20	68
25	77
30	86

FIGURE 3.25 Outcome of Exercise 15.

```
Enter coefficient of restitution: .7
Enter initial height in meters: 8
Number of bounces: 13
Meters traveled: 44.82
```

FIGURE 3.26 Possible outcome of Exercise 16.

16. Bouncing Ball The *coefficient of restitution* of a ball, a number between 0 and 1, specifies how much energy is conserved when the ball hits a rigid surface. A coefficient of .9, for instance, means a bouncing ball will rise to 90% of its previous height after each bounce. Write a program to input a coefficient of restitution and an initial height in meters, and report how many times a ball bounces when dropped from its initial height before it rises to a height of less than 10 centimeters. Also report the total distance traveled by the ball before this point. See Fig. 3.26. The coefficients of restitution of a tennis ball, basketball, super ball, and softball are .7, .75, .9, and .3, respectively.

In Exercises 17 and 18, write a program corresponding to the flowchart.

17. Greatest Common Divisor The flowchart in Fig. 3.29 finds the greatest common divisor (GCD) of two nonzero integers input by the user. ***Note:*** The GCD of two numbers is the largest integer that divides both. See Fig. 3.27.

```
Enter value of M: 30
Enter value of N: 35
Greatest common divisor: 5
```

FIGURE 3.27 Possible outcome of Exercise 17.

```
Enter a positive integer (>1): 2345
Prime factors are 5 7 67
```

FIGURE 3.28 Possible outcome of Exercise 18.

18. Factorization The flowchart in Fig. 3.30 requests a whole number greater than 1 as input and factors it into a product of prime numbers. ***Note:*** A number is *prime* if its only factors are 1 and itself. See Fig. 3.28.

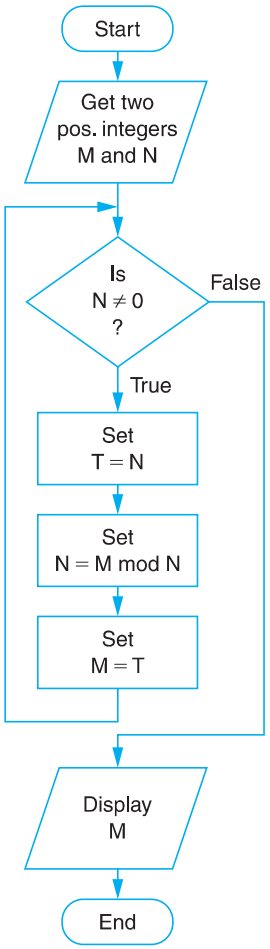

FIGURE 3.29 Greatest common divisor.

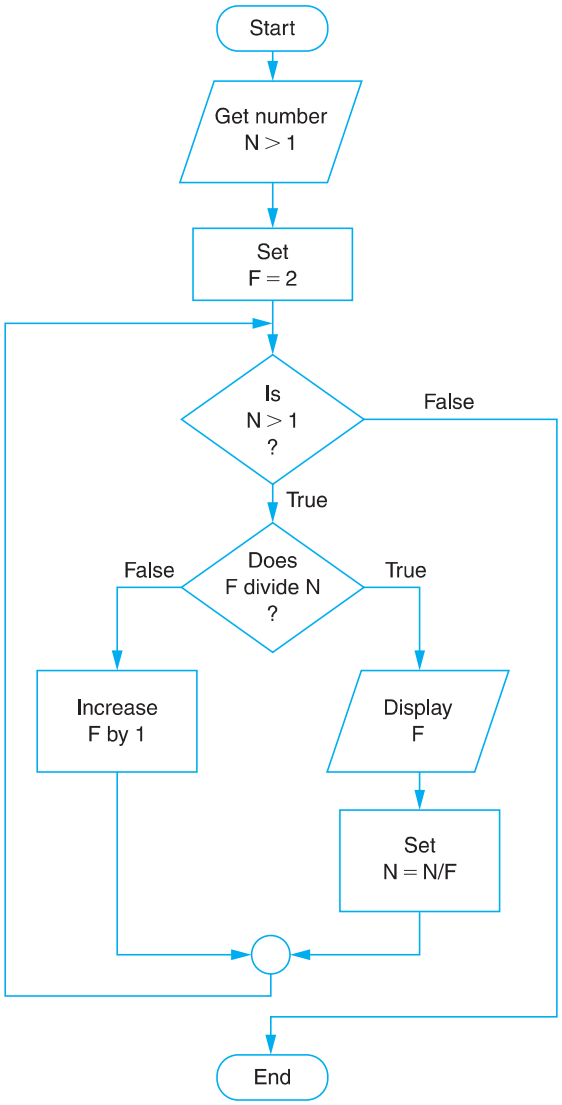

FIGURE 3.30 Prime factors.

In Exercises 19 through 31, write a program to answer the question.

19. Age A person born in 1980 can claim, "I will be x years old in the year x squared." What is the value of x? See Fig. 3.31.

```
Person will be 45
in the year 2025.
```

FIGURE 3.31 Outcome of Exercise 19.

```
World population will be
8 billion in the year 2024.
```

FIGURE 3.32 Outcome of Exercise 20.

20. Population Growth The world population reached 7 billion people on October 21, 2011, and was growing at the rate of 1.1% each year. Assuming that the population continues to grow at the same rate, approximately when will the population reach 8 billion? See Fig. 3.32.

21. Radioactive Decay Strontium-90, a radioactive element that is part of the fallout from nuclear explosions, has a half-life of 28 years. This means that a given quantity of strontium-90 will emit radioactive particles and decay to one-half its size every 28 years. How many years are required for 100 grams of strontium-90 to decay to less than 1 gram? See Fig. 3.33.

```
The decay time is
196 years.
```

FIGURE 3.33 Outcome of Exercise 21.

```
Consumer prices will
double in 29 years.
```

FIGURE 3.34 Outcome of Exercise 22.

22. Consumer Price Index The *consumer price index (CPI)* indicates the average price of a fixed basket of goods and services. It is customarily taken as a measure of inflation and is frequently used to adjust pensions. The CPI was 9.9 in July 1913, was 100 in July 1983, and was 238.25 in July 2014. This means that \$9.90 in July 1913 had the same purchasing power as \$100.00 in July 1983, and the same purchasing power as \$238.25 in July 2014. In 2009, the CPI fell for the first time since 1955. However, for most of the preceding 15 years it had grown at an average rate of 2.5% per year. Assuming that the CPI will rise at 2.5% per year in the future, in how many years will the CPI have at least doubled from its July 2014 level? ***Note:*** Each year, the CPI will be 1.025 times the CPI for the previous year. See Fig. 3.34.

23. Car Loan When you borrow money to buy a house or a car, the loan is paid off with a sequence of equal monthly payments with a stated annual interest rate compounded monthly. The amount borrowed is called the *principal*. If the annual interest rate is 6% (or .06), then the monthly interest rate is .06/12 = .005. At any time, the *balance* of the loan is the amount still owed. The balance at the end of each month is calculated as the balance at the end of the previous month, plus the interest due on that balance, and minus the monthly payment. For instance, with an annual interest rate of 6%,

$$\begin{aligned}[\text{new balance}] &= [\text{previous balance}] + .005 \cdot [\text{previous balance}] - [\text{monthly payment}] \\ &= 1.005 \cdot [\text{previous balance}] - [\text{monthly payment}].\end{aligned}$$

Suppose you borrow \$15,000 to buy a new car at 6% interest compounded monthly and your monthly payment is \$290.00. After how many months will the car be half paid off? That is, after how many months will the balance be less than half the amount borrowed? See Fig. 3.35 on the next page.

```
Loan will be half paid
off after 33 months.
```

FIGURE 3.35 Outcome of Exercise 23.

```
Annuity will be worth more
than $3000 after 29 months.
```

FIGURE 3.36 Outcome of Exercise 24.

24. Annuity An *annuity* is a sequence of equal periodic payments. One type of annuity, called a *savings plan*, consists of monthly payments into a savings account in order to generate money for a future purchase. Suppose you decide to deposit $100 at the end of each month into a savings account paying 3% interest compounded monthly. The monthly interest rate will be .03/12 or .0025, and the balance in the account at the end of each month will be computed as

$$[\text{balance at end of month}] = (1.0025) \cdot [\text{balance at end of previous month}] + 100.$$

After how many months will there be more than $3,000 in the account? See Fig. 3.36.

25. Annuity An *annuity* is a sequence of equal periodic payments. For one type of annuity, a large amount of money is deposited into a bank account and then a fixed amount is withdrawn each month. Suppose you deposit $10,000 into such an account paying 3.6% interest compounded monthly, and then withdraw $600 at the end of each month. The monthly interest rate will be .036/12 or .003, and the balance in the account at the end of each month will be computed as

$$[\text{balance at end of month}] = (1.003) \cdot [\text{balance at end of previous month}] - 600.$$

After how many months will the account contain less than $600, and what will be the amount in the account at that time? See Fig. 3.37.

```
Balance will be $73.91
after 17 months.
```

FIGURE 3.37 Outcome of Exercise 25.

```
Carbon-14 has a half-life
of 5776 years.
```

FIGURE 3.38 Outcome of Exercise 26.

26. Radioactive Decay Carbon-14 is constantly produced in Earth's upper atmosphere due to interactions between cosmic rays and nitrogen, and is found in all plants and animals. After a plant or animal dies, its amount of carbon-14 decreases by about .012% per year. Determine the half-life of carbon-14, that is, the number of years required for 1 gram of carbon-14 to decay to less than ½ gram. See Fig. 3.38.

27. Same Birthday as You Suppose you are in a large-lecture class with n other students. Determine how large n must be such that the probability that someone has the same birthday as you is greater than 50%? See Fig. 3.39. ***Note:*** Forgetting about leap years and so assuming 365 days in a year, the probability that no one has the same birthday as you is $\left(\frac{364}{365}\right)^n$.

```
With 253 students, the
probability is greater than
50% that someone has the same
birthday as you.
```

FIGURE 3.39 Outcome of Exercise 27.

```
Enter amount of deposit: 10000
Balance will be $73.19
after 17 months.
```

FIGURE 3.40 Possible outcome of Exercise 28.

28. Annuity Redo Exercise 25 with the amount of money deposited being input by the user. See Fig. 3.40.

29. Population Growth In 2014 China's population was about 1.37 billion and growing at the rate of .51% per year. In 2014 India's population was about 1.26 billion and growing at the rate of 1.35% per year. Determine when India's population will surpass China's population. Assume that the 2014 growth rates will continue. See Fig. 3.41.

```
India's population will exceed China's
population in the year 2025.
```

FIGURE 3.41 Outcome of Exercise 29.

```
The coffee will cool to below
150 degrees in 7 minutes.
```

FIGURE 3.42 Outcome of Exercise 30.

30. Cooling *Newton's Law of Cooling* states that when a hot liquid is placed in a cool room, each minute the decrease in the temperature is approximately proportional to the difference between the liquid's temperature and the room's temperature. That is, there is a constant k such that each minute the temperature loss is $k \cdot$(liquid's temperature − room's temperature). Suppose a cup of 212°F coffee is placed in a 70°F room and that $k = .079$. Determine the number of minutes required for the coffee to cool to below 150°F. See Fig. 3.42.

31. Saving Account Write a menu-driven program that allows the user to make transactions to a savings account. Assume that the account initially has a balance of $1,000. See Fig. 3.43.

```
Options:
1. Make a Deposit
2. Make a Withdrawal
3. Obtain Balance
4. Quit
Make a selection from the options menu: 1
Enter amount of deposit: 500
Deposit Processed.
Make a selection from the options menu: 2
Enter amount of withdrawal: 2000
Denied. Maximum withdrawal is $1,500.00
Enter amount of withdrawal: 600
Withdrawal Processed.
Make a selection from the options menu: 3
Balance: $900.00
Make a selection from the options menu: 4
```

FIGURE 3.43 Possible outcome of Exercise 31.

Solutions to Practice Problems 3.3

1. initial_val will never change. To correct the program, add the statement initial_val −= 1
2. Either precede the loop with the statement `answer = ""`, or replace with the following loop.

```
while True:
    answer = input("Enter the password: ")
```

```
        if answer.upper() == "SHAZAM":
            break
    print("You may continue.")
```

3. Remove the statement i = i + 1 in the loop, and change the starting index to i + 1.


VideoNote
The *for* Loop

3.4 The *for* Loop

The **for** loop is used to iterate through a sequence of values. The general form of a **for** loop is

```
for var in sequence:
    indented block of statements
```

where *sequence* might be an arithmetic progression of numbers, a string, a list, a tuple, or a file object. The variable is successively assigned each value in the sequence and the indented block of statements is executed after each assignment. Each statement in the block is indented to the same indentation level. This physical indentation tells the interpreter where the block starts and stops.

Looping Through an Arithmetic Progression of Numbers

The range function is used to generate an arithmetic progression of numbers. If m and n have integer values and $m < n$, then the function

```
range(m, n)
```

generates the sequence of integers $m, m + 1, m + 2, \ldots, n - 1$.

That is, the sequence begins with m, and 1 is repeatedly added to m until the number just before n is reached. Some examples are as follows:

`range(3, 10)` generates the sequence 3, 4, 5, 6, 7, 8, 9.

`range(0, 4)` generates the sequence 0, 1, 2, 3.

`range(-4, 2)` generates the sequence −4, −3, −2, −1, 0, 1.

The function `range(0, n)` can be abbreviated to `range(n)` and is usually written in that form.

The loop

```
for num in range(m, n):
    indented block of statements
```

executes the statement(s) in the block once for each integer in the sequence generated by **range(m, n)**. The line beginning with **for** is called the **header** of the loop. The variable following the word **for** is called the **loop variable**, the indented block of statements is called the **body** of the loop, and each execution of the body is referred to as a **pass** through the loop. The header creates the loop variable and successively assigns it the numbers in the sequence, with each assignment followed by a pass through the loop. The most common single-letter names for loop variables are *i*, *j*, and *k*; however, when appropriate, the name should suggest the meaning of the numbers generated. For instance, we might write `for year in range(2000, 2015)`.

Example 1 **Squares** The following two lines of code display four integers and their squares. The loop variable *i* first assumes the value 2 and uses that value in the execution of the **print** statement. The variable then successively assumes and executes the **print** statement for each of the integers 3, 4, and 5.

```
for i in range(2, 6)
    print(i, i * i)
```

[Run]

```
2 4
3 9
4 16
5 25
```

Example 2 **Population Growth** Suppose the population of a city was 300,000 in the year 2014 and is growing at the rate of 3% per year. The following program displays a table showing the population each year until 2018. Figure 3.44 shows the flowchart for the program.

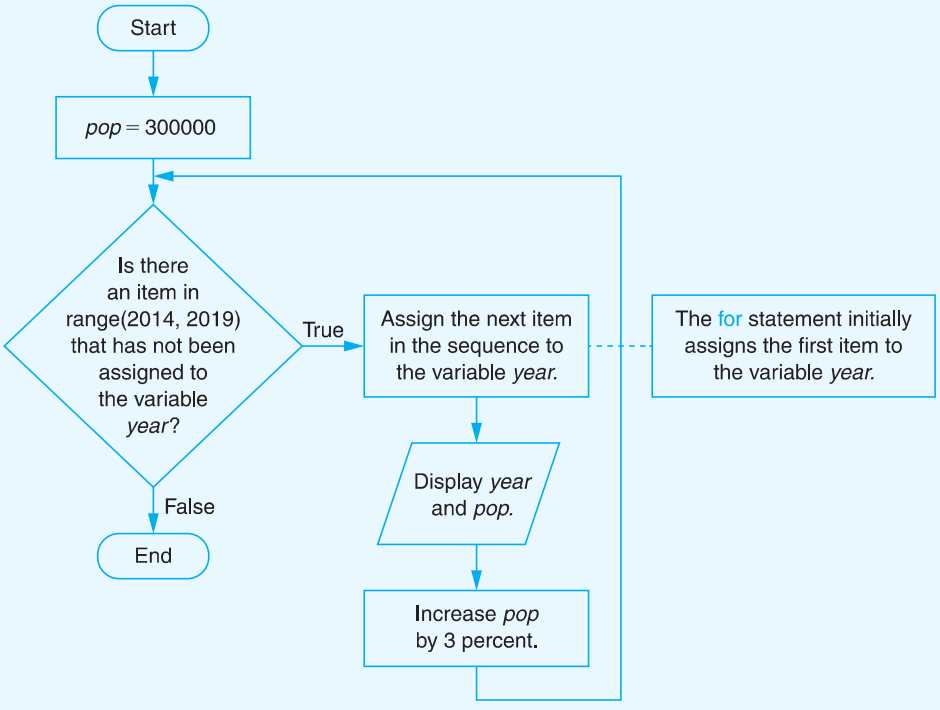

FIGURE 3.44 Flowchart for Example 2.

```
## Display population from 2014 to 2018.
pop = 300000
print("{0:10} {1}".format("Year", "Population"))
```

```
for year in range(2014, 2019):
    print("{0:<10d} {1:,d}".format(year, round(pop)))
    pop += 0.03 * pop     # Increase pop by 3 percent.
```

[Run]

```
Year  Population
2014  300,000
2015  309,000
2016  318,270
2017  327,818
2018  337,653
```

Step Values for the *range* Function

A variation of the range function generates a sequence of integers where successive integers differ by a value other than 1. If m, n, and s have integer values where $m < n$ and s is positive, then the function

range(*m*, *n*, *s*)

generates the sequence of integers m, $m + s$, $m + 2s$, $m + 3s$, . . . , $m + r \cdot s$, where r is the largest whole number for which $m + r \cdot s < n$. That is, the sequence begins with m, and s is repeatedly added to m until the next addition of s would have resulted in a number $\geq n$. The optional number s is called the **step value** of the range function. Some examples are as follows:

range(3, 10, 2) generates the sequence 3, 5, 7, 9.

range(0, 24, 5) generates the sequence 0, 5, 10, 15, 20.

range(-10, 10, 4) generates the sequence −10, −6, −2, 2, 6.

Example 3 Savings Account When A dollars is deposited into an account at the annual interest rate r (in decimal form) compounded monthly, the balance after m months is $A \cdot \left(1 + \frac{r}{12}\right)^m$. The following program requests the amount deposited into a savings account and the annual rate of interest, and then calculates the balance in the account after each quarter-year for four quarters.

```
## Calculate balance in savings account after every three months.
# Obtain input.
initialDeposit = eval(input("Enter amount deposited: "))
prompt = "Enter annual rate of interest; such as .02, .03, or .04: "
annualRateOfInterest = eval(input(prompt))
monthlyRateOfInterest = annualRateOfInterest / 12
# Display table.
print("{0}{1:>15}".format("Month", "Balance"))
for i in range(3, 13, 3):
    print("{0:2}           ${1:<15,.2f}".
        format(i, initialDeposit * (1 + monthlyRateOfInterest) ** i))
```

[Run]

```
Enter amount deposited: 1000
Enter annual rate of interest; such as .02, .03, or .04: .03
Month    Balance
 3       $1,007.52
 6       $1,015.09
 9       $1,022.73
12       $1,030.42
```

In the range functions considered so far, the initial value was less than the terminating value and the step value was positive. However, if a negative step value is used and the initial value is greater than the terminating value, then the range function generates a decreasing sequence that begins with the initial value and decreases until just before reaching the terminating value. Some examples are as follows:

`range(6, 0, -1)` generates the sequence 6, 5, 4, 3, 2, 1.

`range(5, 2, -3)` generates the sequence 5.

`range(10, -10, -4)` generates the sequence 10, 6, 2, −2, −6.

Nested *for* Loops

The body of a for loop can contain any type of Python statement. In particular, it can contain another for loop. However, the second loop must be completely contained inside the first loop and must have a different loop variable. Such a configuration is called **nested for loops**.

Example 4 Multiplication Table The following program displays a multiplication table for the integers from 1 to 5. Here *m* denotes the left factors of the products, and *n* denotes the right factors. Each factor takes on a value from 1 to 5. The values are assigned to *m* in the outer loop and to *n* in the inner loop. Initially, *m* is assigned the value 1, and then the inner loop is traversed five times to produce the first row of products. At the end of these five passes, the value of *m* will still be 1, and the first execution of the inner loop will be complete. Following this, *m* assumes the next number in the sequence, 2. The header of the inner loop is then executed, and resets the value of *n* to 1. The second row of products is displayed during the next pass of the inner loop, and so on.

```
## Display a multiplication table for the numbers from 1 through 5.
for m in range(1, 6):
    for n in range(1, 6):
        print(m, 'x', n, '=', m * n, "\t", end="")
    print()
```

[Run]

```
1 x 1 = 1       1 x 2 = 2       1 x 3 = 3       1 x 4 = 4       1 x 5 = 5
2 x 1 = 2       2 x 2 = 4       2 x 3 = 6       2 x 4 = 8       2 x 5 = 10
3 x 1 = 3       3 x 2 = 6       3 x 3 = 9       3 x 4 = 12      3 x 5 = 15
4 x 1 = 4       4 x 2 = 8       4 x 3 = 12      4 x 4 = 16      4 x 5 = 20
5 x 1 = 5       5 x 2 = 10      5 x 3 = 15      5 x 4 = 20      5 x 5 = 25
```

Example 5 Triangle of Asterisks The following program uses nested `for` loops to display a triangle of asterisks.

```
## Display a triangle of asterisks.
numberOfRows = int(input("Enter a number from 1 through 20: "))
for i in range(numberOfRows):
    for j in range(i + 1):
        print("*", end="")
    print()
```

[Run]

```
Enter a number from 1 through 20: 5
*
**
***
****
*****
```

Looping Through the Characters of a String

If *str1* has a string value, then the loop

```
for ch in str1:
    indented block of statements
```

executes the statement(s) in the body once for each character of the string beginning with the first character. Therefore, there are `len(str1)` passes through the loop.

Example 6 Reverse Letters The following program requests a word as input and displays it backward. The program creates a string consisting of the first letter of the word, and then successively appends each subsequent letter to the front of the string.

```
## Reverse the letters in a word.
word = input("Enter a word: ")
reversedWord = ""
for ch in word:
    reversedWord = ch + reversedWord
print("The reversed word is " + reversedWord + ".")
```

[Run]

```
Enter a word: zeus
The reversed word is suez.
```

Looping Through the Items of a List or Tuple

If *listOrTuple* is a list or a tuple, then the loop

```
for item in listOrTuple:
    indented block of statements
```

executes the statement(s) in the body once for each item of the list or tuple beginning with the first item. Therefore, there are `len(listOrTuple)` passes through the loop.

Example 7 *R Months* The following program displays the months whose names contain the letter *r*.

```
## Display months containing the letter "r".
months = ("January", "February", "March", "April", "May", "June",
          "July", "August", "September", "October", "November", "December")
for month in months:
    if 'r' in month.lower():
        print(month)
```

[Run]

```
January
February
March
April
September
October
November
December
```

The program in Example 6 would work exactly the same if a list was used instead of a tuple. We used a tuple since the sequence of items is fixed and not subject to change. Otherwise, we would have used a list. The program accessed every item of the sequence of months, but did not change any values in the sequence in place. A program that both accesses items and changes values in place not only must use a list, but must iterate over the index values of the list.

Example 8 *Abbreviate Months* The following program replaces the name of each month with its three-letter abbreviation. ***Note:*** For any list, call it *list1*, the last item has index len(*list1*) – 1, and therefore range(len (*list1*)) generates the indices of the list.

```
## Replace each month with its three-letter abbreviation.
months = ["January", "February", "March", "April", "May", "June",
          "July", "August", "September", "October", "November", "December"]
for i in range(len(months)):
    months[i] = months[i][0:3]
print(months)
```

[Run]

```
['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul', 'Aug', 'Sep', 'Oct', 'Nov',
'Dec']
```

Example 9 *Deck of Cards* The following program uses nested for loops with a list of ranks and a list of suits to create a list consisting of the 52 cards in a deck of cards. The cards are assigned their ranks in the outer loop and their suits in the inner loop. The first for statement iterates through the items in *ranks* until every item has been accessed. At each iteration of the outer loop, the second for statement iterates through the items of *suits* until all of those items have been accessed. Each pass of the inner loop appends the name of a

card to the list *deckOfCards*. Figure 3.45 shows a flowchart for the nested **for** loops portion of the program.

```
## Display the names of the 52 cards in a deck of cards.
ranks = ['2', '3', '4', '5', '6', '7', '8', '9',
         "10", "jack", "queen", "king", "ace"]
suits = ["spades", "hearts", "clubs", "diamonds"]
deckOfCards = []  # List to hold the names of the 52 cards in a deck.
# Use nested loops to fill the deckOfCards list.
for rank in ranks:
    for suit in suits:
        deckOfCards.append(rank + " of " + suit)
# Display the 52 cards.
for card in deckOfCards:
    print(card)
```

[Run]

```
2 of spades
2 of hearts
.
.
.
ace of clubs
ace of diamonds
```

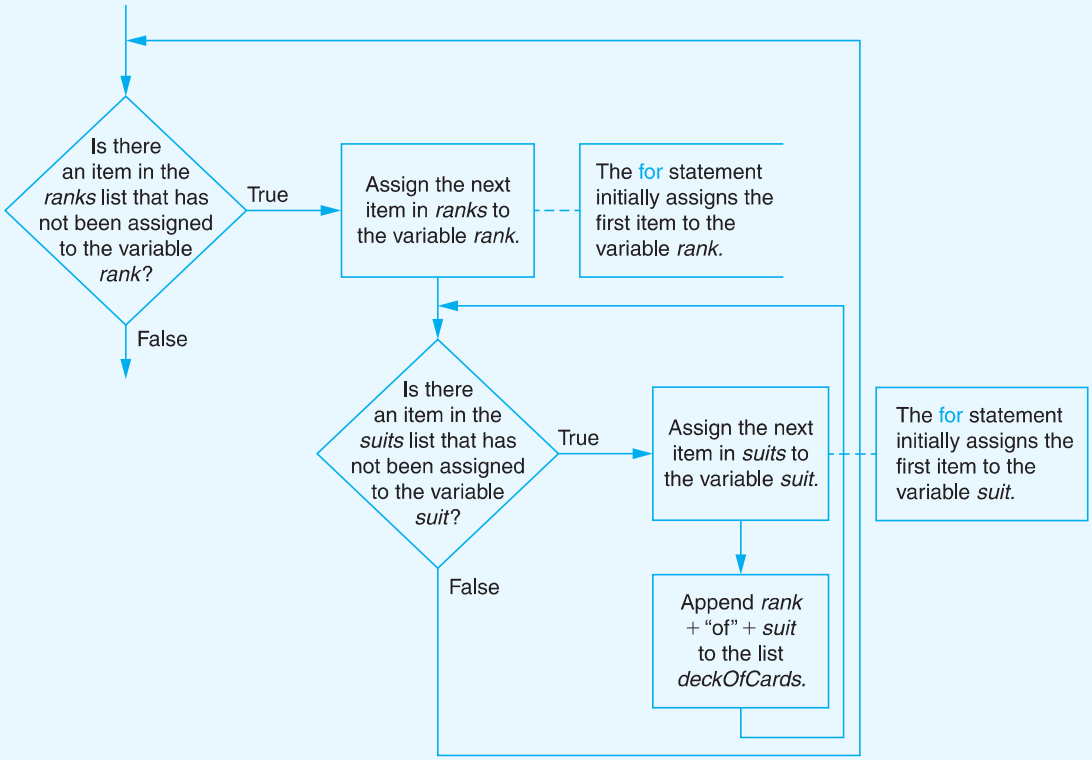

FIGURE 3.45 Flowchart of nested **for** loops for Example 9.

Looping Through the Lines of a Text File

If `fileName.txt` is a text file, then code of the form

```
infile = open("fileName.txt", 'r')
for line in infile:
    indented block of statements
infile.close()
```

reads each line of the file in succession beginning with the first line and executes the indented block of statement(s) for each line. The first statement establishes a connection between the program and the file that allows the program to read data from the file and the last statement terminates the connection. Throughout this textbook, we assume that the file is contained in the same folder as the program file. That way, we can just use the file name in the open function instead of giving a complete path leading to the file.

Example 10 U.S. Presidents The file `USPres.txt` contains the names of the first 44 U.S. presidents in the order in which they served. The following program requests a first name and then displays the names of the U.S. presidents having that first name. The variable *foundFlag* tells us if at least one president had the requested first name. Each line of a text file ends with a special newline character. The rstrip method removes that character.

```
## Display presidents with a specified first name.
firstName = input("Enter a first name: ")
foundFlag = False
infile = open("USPres.txt", 'r')
for line in infile:
    if line.startswith(firstName + ' '):
        print(line.rstrip())
        foundFlag = True
infile.close()
if not foundFlag:
    print("No president had the first name", firstName + '.')
```

[Run]

```
Enter a first name: John
John Adams
John Q. Adams
John Tyler
John Kennedy
```

The *pass* Statement

The header of a for loop must be followed by an indented block of at least one statement. However, there are times when you want the loop to cycle through a sequence and not do anything. In that case, the pass statement should be used. The pass statement is a do-nothing placeholder statement.

Example 11 Last Line of File The following program displays the last line of a file. After the for statement iterates through the entire file, the value of *line* will be the last line of the file. The rstrip method removes the newline character that is at the end of each line of a text file.

```
## Display the last line of a text file.
infile = open("aFile.txt", 'r')
for line in infile:
    pass
print(line.rstrip())
infile.close()
```

Populating a List with the Contents of a Text File

Sometimes the best way to analyze the data in a text file is to place the data into a list and make use of list functions and methods. The following lines of code show one way of placing the contents of a text file into a list.

```
dataList = []
infile = open("Data.txt", 'r')
for line in infile:
    dataList.append(line.strip())
infile.close()
```

However, a more efficient way (to be explained in the next two chapters) is

```
infile = open("Data.txt", 'r')
dataList = [line.rstrip() for line in infile]
infile.close()
```

In either case, each item in the list will be a string. If the file `Data.txt` contained only numbers, the items in the list can be converted from strings to numbers. The following two lines of code will not do the job:

```
for item in dataList:
    item = eval(item)
```

However, the task can be accomplished with

```
for i in range(len(dataList)):
    dataList[i] = eval(dataList[i])
```

A more efficient method (to be explained in the next two chapters) is

```
infile = open("Data.txt", 'r')
dataList = [eval(line) for line in infile]
infile.close()
```

Comments

1. The parentheses of the range function can contain one, two, or three values. When the parentheses contains two or three values, the first value is always the beginning of the sequence generated. When the parentheses contains a single number, call it n, no sequence will be generated when $n \leq 0$; otherwise the sequence of n numbers from 0 to $n - 1$ will be generated.

2. The values generated by the range function can be displayed by applying the list function. For instance, the statement `print(list(range(1, 8, 2)))` displays `[1, 3, 5, 7]`.
3. The function `range(m, n, s)` produces an empty sequence if $n \leq m$ and s is positive or if $m \leq n$ and s is negative.
4. When the statement

   ```
   continue
   ```

   is executed in the body of a for loop, the remaining statements in the body of the loop are skipped and execution continues with the next iteration of the loop.
5. When the statement

   ```
   break
   ```

   is executed in the body of a for loop, the loop is terminated and the loop variable keeps its current value. Both break and continue statements usually appear in the body of an if statement and provide an efficient way of transferring control.
6. *Any* type of loop can be nested inside another loop. For example, for loops can be nested inside while loops and vice versa.
7. The pass statement can be used in any compound statement, such as a while loop or an if-elif-else statement.

Practice Problems 3.4

1. What sequence is generated by the function `range(5)`?
2. Why won't the following lines of code work as intended?

```
for i in range(15, 1):
    print(i)
```

3. Consider the 7th line of Example 10. How would the output change if `print(line.rstrip())` were changed to `print(line)`.
4. Simplify the following code:

```
musketeers = ["Athos", "Porthos", "Aramis", "D'Artagnan"]
i = 0
while i < len(musketeers):
    print(musketeers[i])
    i += 1
```

5. What is the output of the following lines of code?

```
n = 7
for i in range(n):
    print(i, end=" ")
    n = 3
```

EXERCISES 3.4

In Exercises 1 through 8, determine the sequence generated by the range function.

1. `range(1, 10)`
2. `range(1, 10, -1)`
3. `range(10, 1, -1)`
4. `range(5)`

5. `range(5, -1)`
6. `range(-5, 1)`
7. `range(-1, 0)`
8. `range(10, 10)`

In Exercises 9 through 16, determine a range function that generates the sequence of numbers.

9. `4, 9, 14, 19`
10. `0, 1, 2, 3`
11. `-21, -20, -19, -18`
12. `4, 3, 2, 1`
13. `20, 17, 14`
14. `7`
15. `5, 4, 3, 2, 1, 0`
16. `-5, -3, -1, 1`

In Exercises 17 through 40, determine the output displayed.

17.
```
for i in range(1, 5):
    print("Pass #" + str(i))
```

18.
```
for i in range(3, 7):
    print(2 * i)
```

19.
```
num = 5
for i in range(num, 2 * num - 2):
    print(i)
```

20.
```
for i in range(-9, 0, 3):
    print(i)
```

21.
```
# chr(162) is a cents symbol
stringOfCents = ""
for i in range(1, 11):
    stringOfCents += chr(162)
print(stringOfCents)
```

22.
```
n = 3
total = 0
for i in range(1, n + 1):
    total += i
print(total)
```

23.
```
for j in range(2, 9, 2):
    print(j)
print("Who do we appreciate?")
```

24.
```
for countdown in range(10, 0, -1):
    print(countdown)
```

25.
```
number_of_sibilants = 0
word = "stargazers"
for ch in word:
    if (ch == 's') or (ch == 'z'):
        number_of_sibilants += 1
print(number_of_sibilants)
```

26.
```
numCaps = 0
name = "United States of America"
for ch in name:
    if ch.isupper():
        numCaps += 1
print(numCaps)
```

27.
```
word = "183651"
sumOfOddIndexes = 0
oddIndex = False
for ch in word:
    if oddIndex:
        sumOfOddIndexes += int(ch)
    oddIndex = not oddIndex
print(sumOfOddIndexes)
```

28.
```
word = "cloudier"
newWord = ""
evenIndex = True
for ch in word:
    if evenIndex:
        newWord += ch
    evenIndex = not evenIndex
print(newWord)
```

29.
```
for ch in "Python":
    continue
print(ch)
```

30.
```
for ch in "Python":
    break
print(ch)
```

31.
```
numEvens = 0
sumOfEvens = 0
list1 = [2, 9, 6, 7, 12]
```

```
for num in list1:
    if num % 2 == 0:
        numEvens += 1
        sumOfEvens += num
print(numEvens, sumOfEvens)
```

32.
```
list1 = [2, 9, 6, 7, 13, 3]
maxOfOdds = 0
for num in list1:
    if (num % 2 == 1) and (num > maxOfOdds):
        maxOfOdds = num
print(maxOfOdds)
```

33.
```
boroughs = ("Manhatten", "Bronx", "Brooklyn", "Queens", "Staten Island")
minLetters = 100
for borough in boroughs:
    if len(borough) < minLetters:
        minLetters = len(borough)
print("The shortest word has length", minLetters)
```

34.
```
numOfNumbers = 0
list1 = ["three", 4, 5.7, "six", "seven", 8, 3.1416]
for item in list1:
    if isinstance(item, str):
        continue
    numOfNumbers += 1
print(numOfNumbers)
```

35.
```
list1 = [1, 2, "three", 4, 5.7, "six", "seven", 8, 3.1416]
for item in list1:
    if isinstance(item, str):
        break
print(item)
```

36.
```
# I'm looking over a four leaf clover.
leaves = ("sunshine","rain", "the roses that bloom in the lane",
          "somebody I adore")
number = 1
for leaf in leaves:
    print("Leaf", str(number) + ':', leaf)
    number += 1
```

In Exercises 37 and 38, assume that the six lines of the file `Numbers.txt` contain the data 6, 9, 2, 3, 6, and 4.

37.
```
sumEvens = 0
infile = open("Numbers.txt", 'r')
for line in infile:
    if eval(line) % 2 == 0:
        sumEvens += eval(line)
    infile.close()
print(sumEvens)
```

38.
```
dataList = []
infile = open("Numbers.txt", 'r')
```

```
for line in infile:
    dataList.append(eval(line))
infile.close()
print(sum(dataList))
```

In Exercises 39 and 40, assume that the 50 lines of the file `States.txt` contain the names of the fifty states in the order they joined the union.

39.
```
infile = open("States.txt", 'r')
for line in infile:
    if line.startswith("North"):
        print(line, end="")
infile.close()
```

40.
```
infile = open("States.txt", 'r')
for line in infile:
    continue
infile.close()
print(line, end="")
```

In Exercises 41 through 46, identify all errors.

41.
```
for j in range(1, 26, -1):
    print(j)
```

42.
```
for i in range(1, 4):
    print(i + " " + 2 ** i)
```

43.
```
list1 = [2, 5, 7, 2, 7, 8]
list2 = []
for item in list1:
    if item not in list2:
        list2.append(item)
print list2
```

44.
```
list1 = ['a', 'b', 'c']
for letter in list1:
    letter = letter.upper()
print(list1)
```

45.
```
# Display all numbers from 0 through 19 except for 13
for i in range(20, 0):
    if i != 13:
print(i)
```

46.
```
list1 = ["one", "two", "three", "four"]
for item in list1:
    item = item.upper()
print(list1)
```

In Exercises 47 and 48, rewrite the program using a for loop.

47.
```
num = 1
while num <= 9:
    print(num)
    num += 2
```

48.
```
print("hello")
print("hello")
print("hello")
print("hello")
```

Simplify the programs in Exercises 49 and 50.

49.
```
lakes = ["Erie", "Huron", "Michigan", "Ontario", "Superior"]
result = ""
for i in range(len(lakes)):
    result += lakes[i]
    if i < len(lakes) - 1:
        result += ", "
print(result)
```

50.
```
lakes = ["Erie", "Huron", "Michigan", "Ontario", "Superior"]
for i in range(len(lakes)):
    print(lakes[i], end="")
```

```
if i < len(lakes) - 1:
    print(" | ", end="")
```

In Exercises 51 through 65, write a program to carry out the stated task.

51. Radioactive Decay Cobalt-60, a radioactive form of cobalt used in cancer therapy, decays over a period of time. Each year, 12% of the amount present at the beginning of the year will have decayed. If a container of cobalt-60 initially contains 10 grams, determine the amount remaining after five years. Round the amount remaining to two decimal places. See Fig. 3.46.

```
The amount of cobalt-60 remaining
after five years is 5.28 grams.
```

FIGURE 3.46 Outcome of Exercise 51.

```
Enter a telephone number: 982-876-5432
Number without dashes is 9828765432.
```

FIGURE 3.47 Outcome of Exercise 52.

52. Phone Number Remove the dashes from a telephone number input by the user. See Fig. 3.47.

53. Vowels Count the number of vowels in a phrase input by the user. See Fig. 3.48.

```
Enter a phrase: Less is more.
The phrase contains 4 vowels.
```

FIGURE 3.48 Possible outcome of Exercise 53.

```
Enter a number: 3.4
Enter a number: 9.3
Enter a number: 5.5
Largest number: 9.3
```

FIGURE 3.49 Possible outcome of Exercise 54.

54. Largest Number Without using a list, find the largest of three numbers obtained from the user. See Fig. 3.49.

55. Sum of Fractions Find the value of 1 + 1/2 + 1/3 + 1/4 + . . . + 1/100 to five decimal places. See Fig. 3.50.

```
The sum 1 + 1/2 + 1/3 + ... + 1/100
is 5.18738 to five decimal places.
```

FIGURE 3.50 Outcome of Exercise 55.

```
The sum 1 + 2 + ... + 100
is 5050.
```

FIGURE 3.51 Outcome of Exercise 56.

56. Sum of Numbers Find the sum of the first one hundred positive integers. See Fig. 3.51.

57. Alphabetical Order Accept a word as input and determine if its letters are in alphabetical order. Some examples of words whose letters are in alphabetical order are *biopsy*, *adept*, *chintz*, and *lost*. See Fig. 3.52.

```
Enter a word: almost
Letters are in alphabetical order.
```

FIGURE 3.52 Possible outcome of Exercise 57.

```
Enter a word: education
EDUCATION is a vowel word.
```

FIGURE 3.53 Possible outcome of Exercise 58.

58. Vowel Words A **vowel word** is a word that contains every vowel. Some examples of vowel words are *sequoia*, *facetious*, and *dialogue*. Determine if a word input by the user is a vowel word. See Fig. 3.53 on the previous page.

59. Lifetime Earnings Estimate how much a young worker will earn before retiring at age 65, where the worker's name, age, and starting salary are input by the user. Assume the worker receives a 5% raise each year. See Fig. 3.54.

```
Enter name: Helen
Enter age: 25
Enter starting salary: 20000
Helen will earn about $2,415,995.
```

FIGURE 3.54 Possible outcome of Exercise 59.

```
   Simple Interest  Compound Interest
1 $1,050.00        $1,050.00
2 $1,100.00        $1,102.50
3 $1,150.00        $1,157.62
4 $1,200.00        $1,215.51
```

FIGURE 3.55 Outcome of Exercise 60.

60. Simple versus Compound Interest When $1,000 is invested at 5% simple interest, the amount grows by $50 each year. When money is invested at 5% interest compounded annually, the amount at the end of each year is 1.05 times the amount at the beginning of that year. Display the amounts after the first four years for a $1,000 investment at 5% simple and compound interest. See Fig. 3.55.

61. Car Loan Consider the car loan discussed in Exercise 23 of Section 3.3. The loan will be paid off after five years. Assume that the car was purchased at the beginning of January 2013, and display the balance at the end of each year for five years. See Fig. 3.56. ***Note:*** The last payment will be slightly less than the other payments, since otherwise the final balance would be a negative amount.

```
          AMOUNT OWED AT
YEAR      END OF YEAR
2013      $12,347.85
2014      $9,532.13
2015      $6,542.74
2016      $3,368.97
2017      $0.00
```

FIGURE 3.56 Outcome of Exercise 61.

```
          BALANCE AT
YEAR      END OF YEAR
2014      $1,216.64
2015      $2,470.28
2016      $3,762.06
2017      $5,093.12
2018      $6,464.67
```

FIGURE 3.57 Outcome of Exercise 62.

62. Annuity Refer to the annuity discussed in Exercise 24 of Section 3.3. Assume that the first deposit is made at the end of January 2014, and display the balance in the account at the end of each year from 2014 to 2018. See Fig. 3.57.

63. Average Grade Ask the user to enter three grades, and then compute the average after dropping the lowest grade. See Fig. 3.58.

64. Automobile Depreciation A rule of thumb states that cars in personal use depreciate by 15% each year. Suppose a new car is purchased for $20,000. Produce a table showing the value of the car at the end of each of the next four years. See Fig. 3.59.

```
Enter a grade: 70
Enter a grade: 90
Enter a grade: 80
Average: 85
```

FIGURE 3.58 Possible outcome of Exercise 63.

```
1     $17,000.00
2     $14,450.00
3     $12,282.50
4     $10,440.12
```

FIGURE 3.59 Outcome of Exercise 64.

65. Supply and Demand Each year's level of production and price (per bushel) for most agricultural products affects the level of production and price for the following year. Suppose the soybean crop in a country was 80 million bushels in 2014 and

$$[\text{price each year}] = 20 - .1 * [\text{quantity that year}]$$
$$[\text{quantity each year}] = 5 * [\text{price from the preceding year}] - 10,$$

where quantity is measured in units of millions of bushels. Generate a table to show the quantity and price from 2014 until 2018. See Fig. 3.60.

YEAR	QUANTITY	PRICE
2014	80.00	$12.00
2015	50.00	$15.00
2016	65.00	$13.50
2017	57.50	$14.25
2018	61.25	$13.88

FIGURE 3.60 Outcome of Exercise 65.

```
How many numbers do you want to enter? 4
Enter a number: 9
Enter a number: 3
Enter a number: 6
Enter a number: 5
Median: 5.5
```

FIGURE 3.61 Possible outcome of Exercise 66.

66. Median The **median** of an ordered set of measurements is a number separating the lower half from the upper half. If the number of measurements is odd, the median is the middle measurement. If the number of measurements is even, the median is the average of the two middle measurements. Write a program that requests a number n and a set of n measurements (not necessarily ordered) as input and then displays the median of the measurements. See Fig. 3.61.

67. Salary Options Suppose you are given the following two salary options:

Option 1: $20,000 per year, with a raise of $1,000 at the end of each year
Option 2: $10,000 per half-year, with a raise of $250 per half-year at the end of each half-year

Write a program to calculate the amount you would receive for the next ten years under each option to determine the best choice. See Fig. 3.62. (Many people are surprised at the answer.)

```
Option 1 earns $245,000.
Option 2 earns $247,500.
```

FIGURE 3.62 Outcome of Exercise 67.

```
The value of the stock at the
end of the year was $9,483.48.
```

FIGURE 3.63 Outcome of Exercise 68.

68. Misleading Percentages At the beginning of the year you purchased a stock for $10,000. At the end of the year you are told that your stock gained 18% during the past month and that the average monthly change was +1%. Sounds like good news, doesn't it? Later you learn that your stock lost 16% during each of the first six months of the year and gained 18% during each of the last six months of the year. Write a program to determine the value of the stock at the end of the year. See Fig. 3.63.

The file `ColoredBalls.txt` contains the sequence in which colored balls were drawn out of a bag. The file is available on the companion website. Use the file in Exercises 69 and 70.

69. Colored Balls Write a program to determine the number of red balls in the bag. See Fig. 3.64.

```
Number of red balls: 12
```

FIGURE 3.64 Outcome of Exercise 69.

```
A red ball was first drawn in turn 2.
```

FIGURE 3.65 Outcome of Exercise 70.

70. Colored Balls Write a program to determine the first turn in which a red ball was drawn from the bag. See Fig. 3.65.

71. Average Grade The file `Final.txt` contains student grades on a final exam. Write a program that displays the number of grades, the average grade, and the percentage of grades that are above the average grade. See Fig. 3.66.

```
Number of grades: 24
Average grade: 83.25
Percentage of grades above
average: 54.17%
```

FIGURE 3.66 Outcome of Exercise 71.

```
Enter one of five grades: 84
Enter one of five grades: 96
Enter one of five grades: 88
Enter one of five grades: 77
Enter one of five grades: 90
Average grade: 91.33
```

FIGURE 3.67 Possible outcome of Exercise 72.

72. Average Grade Write a program that requests five grades as input and then calculates the average after dropping the two lowest grades. See Fig. 3.67.

73. Number of Vowels Write a program that requests a word as input and counts the number of different vowels in the word. See Fig. 3.68.

```
Enter a word: successful
Number of different vowels: 2
```

FIGURE 3.68 Possible outcome of Exercise 73.

```
Starting word: NAISNIENLGELTETWEORRSD
Crossed out letters: N I N E L E T T E R S
Remaining letters: A S I N G L E W O R D
```

FIGURE 3.69 Outcome of Exercise 74.

74. A Puzzle The following puzzle is known as *The Big Cross-Out Swindle*."Beginning with the word 'NAISNIENLGELTETWEORRSD,' cross out nine letters in such a

way that the remaining letters spell a single word". Write a program that creates variables named *startingWord*, *crossedOutLetters*, and *remainingLetters*. The program should assign to *startingWord* the string given in the puzzle, assign to *crossedOutLetters* a list containing every other letter of *startingWord* beginning with the initial letter *N*, and assign to *remainingLetters* a list containing every other letter of *startingWord* beginning with the second letter, *A*. The program should then display the values of the three variables. See Fig. 3.69.

75. Same Birthday In a group of r people, the probability that at least two people have the same birthday is

$$1 - \left(\frac{n}{n} \times \frac{n-1}{n} \times \frac{n-2}{n} \times \cdots \times \frac{n-(r-1)}{n}\right)$$

where n is the number of days in the year. Write a program that calculates the probabilities for $r = 21$ through 25. Use $n = 365$. See Fig. 3.70.

```
NUMBER OF PEOPLE  PROBABILITY
21                0.444
22                0.476
23                0.507
24                0.538
25                0.569
```

FIGURE 3.70 Outcome of Exercise 75.

```
Connecticut
Delaware
Georgia
Maryland
Massachusetts
```

FIGURE 3.71 Partial Outcome of Exercise 76.

76. Original U.S. States The file `States.txt` contains the 50 U.S. states in the order in which they joined the union. Write a program to display the original 13 states in alphabetical order. Fig. 3.71 shows the first five lines of output.

77. Boston Accent Write a program that asks the user to input a sentence and then displays the sentence with all occurrences of the letter *r* removed. See Fig. 3.72.

```
Enter a sentence: Park the car in Harvard Yard.
Revised sentence: Pak the ca in Havad Yad.
```

FIGURE 3.72 Possible outcome of Exercise 77.

78. Special Number Write a program to find the four-digit number, call it *abcd*, whose digits are reversed when the number is multiplied by 4. That is, $4 \times abcd = dcba$. See Fig. 3.73.

```
Since 4 times 2178 is 8712,
the special number is 2178.
```

FIGURE 3.73 Outcome of Exercise 78.

In Exercises 79 and 80, use the file `ICCWinners.txt` provided on the companion website.

79. ICC Winners Write a program that places the names of the 11 winners of the ICC Cricket World Cup into a list and uses the list to determine the name of the sixth winner. See Fig 3.74.

```
The 6th winner was Sri Lanka.
```

FIGURE 3.74 Outcome of Exercise 79.

```
The 4th winner was Australia.
```

FIGURE 3.75 Outcome of Exercise 80.

80. ICC Winners Write a program that determines the name of the 4th winner of the ICC Cricket World Cup. Do not use a list in the program. See Fig 3.75.

81. Odometer Readings The numbers appearing on a car's odometer range from 000000 to 999999. Write a program to determine the number of readings that contain the digit 1. See Fig. 3.76.

```
468,559 numbers on the odometer
contain the digit 1.
```

FIGURE 3.76 Outcome of Exercise 81.

```
The sum of the digits in the numbers
from 1 to one million is 27,000,001.
```

FIGURE 3.77 Outcome of Exercise 82.

82. Digit Sum Write a program to calculate the total sum of the digits in the integers from 1 to a million. See Fig. 3.77.

83. Flowers and Vegetables The following list contains a mix of names of vegetables and flowers. Each name is followed by a letter about whether it is a vegetable or a flower.

```
mixed = ["Broccoli V", "Lily F", "Cucumber V", "Rose F", "Lotus
F", "Cabbage V", "Onion V", "Anemone F", "Aster F"]
```

Write a program that creates two lists (one of vegetables, and one of flowers) and uses the lists to produce the output shown in Fig 3.78.

```
Vegetables: Broccoli, Cucumber, Cabbage, Onion.
Flowers: Lily, Rose, Lotus, Anemone, Aster
```

FIGURE 3.78 Outcome of Exercise 83.

Solutions to Practice Problems 3.4

1. The sequence generated by `range(5)` is the same as the sequence generated by `range(0, 5)`, that is, 0, 1, 2, 3, 4.
2. The loop will never be entered because 15 is greater than 1. The intended first line might have been

   ```
   for i in range (15, 1, -1):
   ```

   or

   ```
   for i in range(2, 16):
   ```

3. Since each line of the file ends with a newline character, the names would have been displayed double-spaced.
4.
   ```
   musketeers = ["Athos", "Porthos", "Aramis", "D'Artagnan"]
   for name in musketeers:
       print(name)
   ```

   When no items of a list will be altered in place, use in instead of indices to iterate through the list.
5. 0 1 2 3 4 5 6. The header of the for loop generates a sequence of 7 numbers to be iterated over. That sequence is permanent. There will be 7 passes through the loop unless a break statement is encountered.

CHAPTER 3 KEY TERMS AND CONCEPTS	EXAMPLES
3.1 Relational and Logical Operators	
The **Boolean data type** (or **bool**) has the two values True and False.	
The **ASCII table** associates characters with nonnegative numbers. The value of **chr(n)** is the character associated with the number *n*. The function **ord** is the inverse of the **chr** function. The use of the ASCII table to order items of data is called **lexicographical ordering**.	chr(49) is '1', chr(65) is 'A', chr(97) is 'a'. ord('9') is 57, ord('Z') is 90, ord('z') is 122. "Spam" < "spa" lexicographically. [8, 'X'] < [8, 'x'] has the value True.
The **relational operators** are <, >, ==, !=, <=, >=, *in*, and *not in*.	2 < 3, 2 != 3, 'a' in ['a', 'b'] have value True. 2 == 3, "spam"<="Spam" have value False.
The principal **logical operators** are *and*, *or*, and *not*.	(7 < 5) or (2 != 3) has value True.
A **condition** is an expression involving literals, variables, functions, and operators (arithmetic, relational, or logical) that can be evaluated as True or False.	Let a = 3 and b = "spam". ((5 < (2*a)) and (len(b) == 4)) has value True; not ((2*len(b)) == 8) has value False.
The list **sort method** lexicographically orders items.	['b', 'a', 'c'].sort() is ['a', 'b', 'c'].
The **startswith** and **endswith** methods return the values that their names imply.	"spam".startswith("sp") has value True. "spam".endswith('m') has value True.
String methods that return Boolean values: isdigit, isalpha, isalnum, islower, isupper, isspace	"hi!".isalpha() has value False. "ne1".isalnum() has value True.
The **in operator** can be used to simplify complex conditions.	(s == 'p') or (s == 'i') or (s == 'e') can be replaced with (s in "pie").
3.2 Decision Structures	

An **if statement** has the form
`if condition1:`
`    indented block of statements`
`elif condition2:`
`    indented block of statements`
`else:`
`    indented block of statements`
It executes the first block whose associated condition is true. The **elif** and **else** clauses are optional and there may be multiple **elif** clauses.
	`n = int(input("Enter an int: "))`
	`if n <= 0:`
	`    print(n, "is neg or zero")`
	`elif n % 2 == 0:`
	`    print(n, "is pos and even")`
	`else:`
	`    print(n, "is pos and odd")`
	[Run]
	`Enter an int: 5`
	`5 is pos and odd`

CHAPTER 3 KEY TERMS AND CONCEPTS	EXAMPLES
3.3 The *while* Loop	

A **while loop** has the form
`while condition:`
`    indented block of statements`
 	
The loop repeatedly executes the block as long as the condition is true.
	`n = 1`
	`while n < 6:`
	`    print(n, end=" ")`
	`    n += 1`
	
	[Run]
	`1 2 3 4 5`

A while loop might use a **counter variable** to keep track of the number of times a certain event has occurred, an **accumulator variable** to hold a total, and a **sentinel value** to indicate the end of a sequence of inputs.	`print("Enter -1 to end input.")`
	`# -1 is the sentinel value`
	`counter = 0`
	`accumulator = 0`
	`s = "Enter a positive integer: "`
	`n = int(input(s))`
	`while n != -1:`
	`    counter += 1`
	`    accumulator += n`
	`    n = int(input(s))`
	`print(counter, "ints entered")`
	`print("Sum:", accumulator)`
	
	[Run]
	`Enter -1 to end input.`
	`Enter a positive integer: 3`
	`Enter a positive integer: 4`
	`Enter a positive integer: -1`
	`2 ints entered`
	`Sum: 7`

If a **break statement** is encountered during a pass through a while loop, the loop is immediately exited.	Can replace lines 6–10 with
	`while True:`
	`    n = int(input(s))`
	`    if n == -1:`
	`        break`
	`    counter += 1`
	`    accumulator += n`

If a **continue statement** is encountered in the block of a while loop, execution jumps back to the closest enclosing while loop header.	`n = 9`
	`while(n <= 15):`
	`    n += 1`
	`# skip unlucky number`
	`    if n == 13:`
	`        continue`
	`    print(n, end=" ")`
	
	[Run]
	`10 11 12 14 15 16`

CHAPTER 3 KEY TERMS AND CONCEPTS	EXAMPLES
3.4 The *for* Loop	
The **range function** generates an arithmetic progression of numbers.	range(5) generates 0, 1, 2, 3, 4 range(1, 9, 2) generates 1, 3, 5, 7

A **for loop** repeats a block of statements as its loop variable iterates through a sequence. The sequence can be an arithmetic progression, the items of a list or tuple, the characters of a string, or the lines of a file object. The statements **break** and **continue** have the same effect in **for** loops as they do in **while** loops.	`list1 = []`
	`for i in range(9, 0, -1):`
	`  list1.append(i)`
	`for item in list1:`
	`  print(item, end="")`
	[Run]
	`987654321`

A **flag** is a variable used to indicate whether a certain event has occurred or a certain situation exists.	See Example 10.
The **pass statement** is a do-nothing placeholder that is sometimes used where the syntax requires a statement.	See Example 11.

CHAPTER 3 PROGRAMMING PROJECTS

1. Car Loan Write a program to analyze a car loan. See Fig. 3.79. The user should enter the amount of the loan, the annual percentage rate of interest, and the duration of the loan in months. After each piece of data is entered, the data should be checked to make sure it is reasonable. If bad data has been supplied, the user should be so advised. Otherwise, the monthly payment and the total amount of interest paid should be displayed. The formula for the monthly payment is

$$\text{monthly payment} = \frac{p \cdot r}{1 - (1 + r)^{-n}},$$

where p is the amount of the loan, r is the monthly interest rate (annual rate divided by 12) given as a number between 0 (for 0%) and 100 (for 100%), and n is the duration of the loan in months. The formula for the total interest paid is

$$\text{total interest} = n \cdot [\text{monthly payment}] - p.$$

```
Enter the amount of the loan: 18000
Enter the interest rate: 5.25
Enter the duration in months: 60
Monthly payment: $341.75
Total interest paid: $2,505.00
```

FIGURE 3.79 Possible outcome of Programming Project 1.

```
Enter a: 1
Enter b: -11
Enter c: 28
Solutions: 7 and 4
```

FIGURE 3.80 Possible outcome of Programming Project 2.

2. Quadratic Equation Write a program to determine the real roots of the quadratic equation $ax^2 + bx + c = 0$ (where $a \neq 0$) after requesting the values of a, b, and c. Before finding the roots, ensure that a is nonzero. [***Note:*** The equation has 2, 1, or 0 solutions depending on whether the value of $b^2 - 4ac$ is positive, zero, or negative. In the first two cases, the solutions are given by the quadratic formula $(-b \pm (b^2 - 4ac)^{.5})/2a$.] See Fig. 3.80 on the previous page.

3. Caffeine Absorption After caffeine is absorbed into the body, 13% is eliminated from the body each hour. Assume a person drinks an 8-oz cup of brewed coffee containing 130 mg of caffeine, and that the caffeine is absorbed immediately into the body. Write a program to calculate the following values. See Fig. 3.81.

(a) The number of hours required until less than 65 mg (one-half the original amount) remain in the body.

(b) The amount of caffeine in the body 24 hours after the person drinks the coffee.

(c) Suppose the person drinks a cup of coffee at 7 a.m. and then drinks a cup of coffee at the end of each hour until 7 a.m. the next day. How much caffeine will be in the body at the end of the 24 hours?

```
CAFFEINE VALUES
One cup: less than 65 mg. will remain after 5 hours.
One cup: 4.60 mg. will remain after 24 hours.
Hourly cups: 969.24 mg. will remain after 24 hours.
```

FIGURE 3.81 Outcome of Programming Project 3.

4. Rule of 72 This rule is used to approximate the time required for prices to double due to inflation. If the inflation rate is r%, then the Rule of 72 estimates that prices will double in $72/r$ years. For instance, at an inflation rate of 6%, prices double in about 72/6 or 12 years. Write a program to test the accuracy of this rule. For each interest rate from 1% to 20%, the program should display the rounded value of $72/r$ and the actual number of years required for prices to double at an r% inflation rate. (Assume prices increase at the end of each year.) Fig. 3.82 shows the first five sets of values.

Interest Rate	Rule of 72 Doubling Time (in years)	Actual Doubling Time (in years)
1%	72	70
2%	36	36
3%	24	24
4%	18	18
5%	14	15

FIGURE 3.82 Partial Outcome of Programming Project 4.

5. Individual Retirement Accounts Money earned in an ordinary savings account is subject to federal, state, and local income taxes. However, a special type of retirement savings account, called a **traditional individual retirement account** (traditional IRA), allows these taxes to be deferred until after retirement. IRAs are highly touted by financial planners. The purpose of this programming project is to show the value of starting an IRA early. Earl and Larry each begin full-time jobs in January 2015 and

plan to retire in January 2063 after working for 48 years. Assume that any money they deposit into IRAs earns 4% interest compounded annually. Earl opens a traditional IRA account immediately and deposits $5,000 into his account at the end of each year for fifteen years. After that he plans to make no further deposits and just let the money earn interest. Larry plans to wait fifteen years before opening his traditional IRA and then deposit $5,000 into the account at the end of each year until he retires. Write a program that calculates the amount of money each person has deposited into his account and the amount of money in each account upon retirement. See Fig. 3.83.

```
                AMOUNTS DEPOSITED
Earl: $75,000.00              Larry: $165,000.00
        AMOUNTS IN IRA UPON RETIREMENT
Earl: $365,268.39             Larry: $331,047.64
```

FIGURE 3.83 **Outcome of Programming Project 5.**

6. Soundex System Soundex is a system that encodes a word into a letter followed by three numbers that roughly describe how the word sounds. Similar sounding words have the same four-character codes. For instance, the words Carrot and Caret are both coded as C123. A slight variation of the Soundex coding algorithm is as follows:

1. Retain the first letter.
2. For the remaining letters, delete all occurrences of *a*, *e*, *i*, *o*, *u*, *h*, *y*, and *w*.
3. Assign numbers to the other letters that remain so that
   - **(a)** *b*, *f*, *p*, and *v* become 1
   - **(b)** *c*, *g*, *j*, *k*, *q*, *s*, *x*, and *z* become 2
   - **(c)** *d* and *t* both become 3
   - **(d)** *l* (i.e., *el*) becomes 4
   - **(e)** *m* and *n* become 5
   - **(f)** *r* becomes 6
4. If two or more letters that have been replaced by the same number were next to each other in the original full word, keep only the first of them.
5. Keep only the first four characters of what you have left. If you have fewer than four, then add zeros on the end to make the string have length four.

Write a program that carries out the algorithm. See Fig. 3.84.

```
Enter a word to code: Robert
The coded word is R163.
```

FIGURE 3.84 **Possible outcome of Programming Project 6.**

7. Error Detection Suppose you type a 14-digit credit card number into a Web site, but mistype one of the digits or inadvertently interchange two adjacent digits. The Web site will perform a validation check that always detects the first type of error and nearly always detects the second type of error. The validation check is as follows:

1. Starting with the leftmost digit, double it and then double every other digit after it. However, if any of the doubled digits is a two-digit number, subtract 9 from it. Then sum these new digits. For instance, if the credit card number is 58667936100244, then the digits considered are 5, 6, 7, 3, 1, 0, 4, their new replacements are 1, 3, 5, 6, 2, 0, 8, and the sum of the replacements is 25.

2. Sum together the remaining seven digits from the credit card number. That is, the digits in the odd-numbered positions. With the credit card number above, we obtain 8 + 6 + 9 + 6 + 0 + 2 + 4 = 35.
3. Add together the two sums. If the result is a multiple of 10, then accept the credit card number. Otherwise, reject it. We accept the credit card number above since 25 + 35 = 60, a multiple of 10.

Write a program that performs data validation on a credit card number. See Fig. 3.85.

```
Enter a credit card number: 58667936100244
The number is valid.
```

FIGURE 3.85 **Possible outcome of Programming Project 7.**

8. Palindrome A *palindrome* is a word or phrase that reads the same forward and backward, character for character, disregarding punctuation, case, and spaces. Some examples are "racecar", "Madam, I'm Adam", and "Was it a cat I saw?". Write a program that asks the user to input a word or phrase and then determines if it is a palindrome. See Fig. 3.86. ***Note:*** Remove all spaces and punctuation before analyzing the word or phrase.

```
Enter a word or phrase: A man, a plan, a canal: Panama.
A MAN, A PLAN, A CANAL: PANAMA. is a palindrome.
```

FIGURE 3.86 **Possible outcome of Programming Project 8.**